

VISIT...

LANZAROTE
Caliente.COM

В. А. Ильина, П. К. Силаев

ЧИСЛЕННЫЕ МЕТОДЫ ДЛЯ ФИЗИКОВ-ТЕОРЕТИКОВ

II



Москва ♦ Ижевск

2004

Интернет-магазин

MAFFESS

<http://shop.rcd.ru>

- физика
 - математика
 - биология
 - нефтегазовые технологии
-

Ильина В. А., Силаев П. К.

Численные методы для физиков-теоретиков. II. — Москва-Ижевск: Институт компьютерных исследований, 2004, 118 стр.

Данное пособие основано на лекциях и практических занятиях по курсу численных методов для будущих физиков-теоретиков. Основная цель книги состоит в рассмотрении понятных и достаточно простых в написании алгоритмов, ориентированных главным образом на решение типичных задач теоретической физики и являющихся, безусловно, необходимой частью арсенала любого физика-теоретика.

Для студентов физических специальностей.

ISBN 5-93972-320-9

© В. А. Ильина, П. К. Силаев, 2004

© Институт компьютерных исследований, 2004

<http://ics.org.ru>

<http://rcd.ru>

Оглавление

1. Предисловие	5
2. Элементарные сведения о параллельных вычислениях	5
2.1. Некоторые общие сведения о параллельных вычислениях	6
2.2. Простейшие сведения о протоколе MPI	9
3. Задача Коши для обыкновенных дифференциальных уравнений	16
3.1. Методы Рунге-Кутты	17
3.1.1. Разнообразные n-точечные схемы	17
3.1.2. Адаптивное изменение шага	21
3.2. Интерполяционные методы	27
3.2.1. Простейшие рецепты	27
3.2.2. Переменный шаг	30
3.3. Простейшие методы «предсказание-коррекция»	32
3.3.1. Схемы Адамса	33
3.3.2. Переменный шаг	35
4. Краевая задача для обыкновенных дифференциальных уравнений	37
4.1. Метод «стрельбы»	38
4.1.1. Общий рецепт	38
4.1.2. Уравнение типа Шредингера	40
4.1.3. Особые точки	45
4.2. Релаксационные методы	47
4.2.1. Метод Ньютона	48
4.2.2. Минимизация	52
4.2.3. Переменный шаг	56
5. «Жесткие» системы	58
5.1. Неявные схемы Рунге-Кутты	63
5.2. Неявные интерполяционные схемы	66
6. Дифференциальные уравнения в частных производных	68
6.1. Задача Коши для линейных уравнений	68
6.2. Задача Коши для гиперболических уравнений	78
6.2.1. Самый «наивный» вариант	78
6.2.2. Простейший анализ устойчивости	79
6.2.3. Простейший удовлетворительный рецепт	81

6.2.4.	Большее число измерений	82
6.3.	Задача Коши для параболических уравнений	85
6.3.1.	Анализ устойчивости	86
6.3.2.	Два простейших рецепта	87
6.3.3.	Большее число измерений	89
6.4.	Краевая задача для линейных уравнений	90
6.5.	Краевая задача для эллиптических уравнений	93
6.5.1.	Чебышевское ускорение	93
6.5.2.	Минимизация	97
6.5.3.	Мультирешетки	98
7.	Интегральные уравнения. Нелокальные уравнения	102
7.1.	Уравнение Вольтерра	104
7.2.	Задача на собственные значения	105
7.3.	Уравнение Фредгольма II рода	106
7.4.	Итерационный метод	107
8.	Некорректные задачи	108
8.1.	«Примитивный» рецепт	109
8.2.	Регуляризация	110
9.	Задачи для вычислительного практикума	112
	Литература	117

1. Предисловие

Пособие представляет собой вторую часть курса численных методов, читаемых студентам кафедры квантовой теории и физики высоких энергий физического факультета МГУ. Все необходимые ссылки на первую часть курса снабжаются префиксом «часть I».

В отборе материала мы придерживались тех же принципов, что и при составлении первой части курса, т. е. старались создать минимальный набор простых в написании, но достаточно эффективных вычислительных рецептов, которые необходимы физику-теоретику. При этом, выбирая между простотой и эффективностью, мы почти всегда выбирали простоту. Так что слова «минимальный» и «простейший» особенно относятся ко второй части курса. Кроме того, как и в первой части, все изложение ведется на физическом уровне строгости, т. е. абсолютно некорректно с математической точки зрения.

Наконец, следует помнить, что, оценивая эффективность тех или иных методов и давая соответствующие рекомендации, авторы опирались на опыт решения тех задач, которые чаще всего встречались им в практике. Так что все эти рекомендации ни в коем случае не являются абсолютными.

2. Элементарные сведения о параллельных вычислениях

Содержание этого раздела является довольно существенным отступлением от идеологии всего пособия. В большинстве случаев мы излагали численные методы как таковые, почти не ссылаясь на особенности конкретного типа компьютеров, операционной системы и конкретного языка (хотя, разумеется, в основном подразумевается язык «C» и операционная система UNIX, живущая на персональных компьютерах типа IBM PC или рабочих станциях DEC или Sun).

В этом разделе мы изложим не столько алгоритмы и методы, сколько простейшие сведения о вполне конкретной реализации параллельных вычислений в рамках протокола MPI. Несмотря на свое довольно скептическое отношение к полезности проекта GRID для отдельно взятого теоретика,

использующего численный счет, авторы уверены, что спустя несколько лет содержимое этого раздела безнадежно устареет и будет представлять лишь исторический интерес. Некоторым оправданием нам может служить то обстоятельство, что протокол MPI реализован не только на «суперкомпьютерах для бедных» (обычных персональных компьютерах, объединенных в компьютерный кластер локальной сетью), но и на «настоящих» многопроцессорных суперкомпьютерах, и, следовательно, просуществует еще некоторое время.

Следует подчеркнуть, что авторы не являются специалистами ни по суперкомпьютерам, ни по параллельным вычислениям. Мы излагаем точку зрения на параллельный счет, которая сформировалась при его реализации для решения конкретных задач на конкретном компьютерном кластере. Авторам известно, что их точка зрения вызывает неудовольствие специалистов по параллельным вычислениям.

2.1. Некоторые общие сведения о параллельных вычислениях

Если выражаться не слишком аккуратно, то можно сказать, что в мире многопроцессорных компьютеров существуют две идеологии: либо компьютер строится на основе небольшого числа очень хороших процессоров (ранние суперкомпьютеры, многопроцессорные Sun'ы), либо, напротив, используется очень большое число не слишком выдающихся процессоров (IBM-овские и Hewlett-Packard-овские суперкомпьютеры).

Довольно ясно, что в первом случае есть (хотя бы принципиальная) возможность написать «родной» для данной машины компилятор, который будет учитывать фактическое количество процессоров и распределять отдельные действия программы по разным процессорам, т.е. реализовывать параллельный счет для обыкновенной («непараллельной») программы.

При всех стандартных оговорках:

- в автоматическом распараллеливании есть подводные камни. Например, если при вычислении суммы знакопеременного ряда четные члены будут суммироваться одним процессором, а нечетные — другим, и только в конце эти два результата сложатся, то мы получим не ответ, а ошибки округления;
- автоматическое распараллеливание на 2 процессора сделать легко, на 4 — возможно, на 8 — сложно, на 16 — почти невозможно,

с точки зрения пользователя такой подход, разумеется, предпочтителен: «пусть машина делает все сама».

Напротив, во втором случае пользователь должен сам написать распараллеленную программу. Распараллеленная программа, запущенная одновременно на нескольких процессорах, должна, во-первых, сообразить, на скольких именно процессорах она запущена (в этих компьютерах есть возможность задействовать под данную задачу не все, а только некоторые процессоры), во-вторых, правильно поделить кусочки задачи между задействованными процессорами и, в-третьих, обеспечить обмен данными и промежуточными ответами между задействованными процессорами.

Следует подчеркнуть, что для каждой конкретной задачи существует своя методика распараллеливания:

- Если задача «естественным образом» разрезается на равноправные и связанные только в пограничных областях куски (типичный пример — описание эволюции в решеточной модели без ярко выраженной нелокальности, т.е. эволюция данного узла решетки определяется только ближайшими соседними узлами), то поделить ее между процессорами довольно просто. Достаточно перенумеровать все задействованные процессоры ($1 \dots N$); поделить решетку на N кусков так, чтобы эти куски слегка перекрывались; и отдать каждому процессору свой кусок решетки. На каждом обороте алгоритма k -й процессор считает эволюцию k -го куска, причем правильные ответы он может получить только во внутренних узлах k -го куска решетки, правильную эволюцию пограничных узлов он сосчитать не может. Это связано с тем, что эволюция левых пограничных узлов k -го куска решетки определяется соседними узлами, которые принадлежат $(k - 1)$ -му куску, а k -й процессор ничего про $(k - 1)$ -й кусок решетки не знает. То же самое происходит на правой границе k -го куска. Поэтому перекрытие кусков должно быть таким, чтобы левые пограничные узлы k -го куска были внутренними узлами $(k - 1)$ -го куска, а правые пограничные узлы k -го куска были внутренними узлами $(k + 1)$ -го куска. Тогда $(k - 1)$ -й процессор сообщит k -му процессору правильные новые значения для левых пограничных узлов, а $(k + 1)$ -й процессор — для правых.
- Если задача в целом не нарезается на кусочки, но по ходу решения есть большие объемы действий, которые можно провести независимо (например, на каждом обороте итерационного процесса надо найти многомерный корень, вычислить интеграл и подействовать матрицей на вектор, а потом собрать все это вместе), то простейшим решением будет «назначить» главный процессор. Этот главный процессор будет раздавать задания в зависимости от числа задействованных процессоров (например, главный процессор ищет корень, второй процессор

вычисляет первую половинку интеграла, третий — вторую половинку интеграла, четвертый и пятый вычисляют первую и вторую половинки ответа для умножения матрицы на вектор). Кроме того, главный процессор будет собирать ответы, полученные «подчиненными».

- Наконец, есть задачи, в которых лучше и вовсе отказаться от распараллеливания. Разумеется, распараллелить можно любую задачу, но иногда потраченные на это усилия не оправдываются.

Язык, на котором процессоры разговаривают между собой, вполне может быть стандартизован, причем вне зависимости от типа процессоров и типа связи между ними. Поскольку есть «естественные потребности» параллельного счета, необходимо реализовать, во-первых, обмен данных между процессорами, во-вторых, обеспечить синхронизацию работы процессоров. Поэтому были созданы протоколы, обеспечивающие обмен информацией между процессорами и координированность их работы. Одним из таких протоколов и является протокол **MPI** (Message-Passing Interface).

Протокол MPI достаточно абстрактен, так что он вполне подходит и для «суперкомпьютера для бедных» — компьютерного кластера, т. е. нескольких (нередко разнотипных) компьютеров, объединенных в локальную сеть. Протокол MPI для компьютерного кластера, называемый «mpich», был реализован в рамках проекта GNU.

Mpich можно найти на сайте

<http://www.mcs.anl.gov>.

На данный момент он лежит на страничке

<http://www-unix.mcs.anl.gov/mpi/mpich>.

Его также можно скачать по anonymous ftp:

<ftp://ftp.mcs.anl.gov/pub/mpi>.

Кроме того, mpich удобно скачивать с многочисленных зеркал, которые легко найти, запустив поиск строки `mpich.tar.gz` в Интернете.

Собирают mpich на базе gcc (GNU C), например, под Linux¹, при этом сооружается свой собственный компилятор mpicc вместо стандартного gcc. В соответствии с принятым на данном кластере режимом доступа с одной машины на другую² создается скрипт для запуска программ на

¹В принципе, mpich существует и для WinNT, но всерьез относиться к численному счету в среде Windows авторы не рекомендуют.

²Если кластер отцеплен от внешнего мира, то разумнее всего использовать беспарольный rsh. К сожалению, обычно в кластерах приходится разрешать доступ только через ssh с паролем. Для работы с mpich это не очень удобно: при каждом запуске программы приходится несколько раз вводить пароль.

нескольких машинах `mpirun`. Кроме того, в конфигурации `mpich` надо прописать имена (`host names`) машин, которые будут задействованы при параллельном счете.

После установки `mpich` пользователь должен написать распараллеленную в соответствии со стандартами MPI программу, откомпилировать ее с помощью `mpicc`, выложить исполняемый файл на все машины, которые он хочет задействовать, и запустить с одной из машин скрипт `mpirun`. После этого (в случае отсутствия ошибок в программе) на кластере начнется параллельный счет.

Возникает вопрос, стоит ли всем этим заниматься?

На наш взгляд, ответ достаточно прост: избегайте параллельного счета всегда, когда это только возможно. Типичная счетная программа всегда содержит достаточное количество ошибок (как чисто технических, так и идеологических), и увеличивать их количество за счет распараллеливания нет никакой необходимости. Кроме того, в реальных задачах довольно часто необходимо выполнять счет с помощью одной и той же программы, но при разных значениях параметров. При наличии кластера естественным решением является не параллельный счет, а запуск задачи с разными параметрами на разных компьютерах.

Параллельного счета невозможно избежать в двух случаях. Во-первых, если задача не помещается в память одной машины. Под памятью здесь имеется в виду фактическая оперативная память. Использование «теоретической» оперативной памяти, т.е. фактической памяти и места на диске (`swap`), обычно означает невозможность сосчитать задачу за конечное время. Во-вторых, параллельного счета невозможно избежать, если задача считается немыслимо долго. Есть существенная разница между неделями счета на 12-и машинах и 10-ю неделями счета на 1-й машине.

2.2. Простейшие сведения о протоколе MPI

Приведем сначала пример простейшей и абсолютно бессмысленной программы, использующей протокол MPI, а затем дадим необходимые пояснения.

```
#include <mpi.h>
#include <stdio.h>
#include <math.h>

main(int npar, char ** par)
```

```
{
int mynum, totnum, len, dat, totsum, totmax;
MPI_Status stat;
char name[MPI_MAX_PROCESSOR_NAME];
double t1, t2;

MPI_Init(&npar, &par);
MPI_Comm_size(MPI_COMM_WORLD, &totnum);
MPI_Comm_rank(MPI_COMM_WORLD, &mynum);
MPI_Get_processor_name(name, &len);

t1 = MPI_Wtime();

if(mynum==0)
    fprintf(stderr, "Total processors number: %d\n",
                                                    totnum);
    fprintf(stderr, "Starting process N %d on %s\n",
                                                    mynum, name);

if(mynum==0) dat=0;
if(mynum>0)
    MPI_Recv(&dat, 1, MPI_INT, mynum-1, 333,
            MPI_COMM_WORLD, &stat);

dat++;
if(mynum<totnum-1)
    MPI_Send(&dat, 1, MPI_INT, mynum+1, 333,
            MPI_COMM_WORLD);
    MPI_Reduce(&dat, &totsum, 1, MPI_INT, MPI_SUM, 0,
            MPI_COMM_WORLD);
    MPI_Reduce(&dat, &totmax, 1, MPI_INT, MPI_MAX, 0,
            MPI_COMM_WORLD);

t2 = MPI_Wtime();

if(mynum==0)
    fprintf(stderr, "Total sum: %d, max: %d,
            time: %.2lf\n", totsum, totmax, t2-t1);

MPI_Finalize();
}
```

Эту программу следует откомпилировать:

```
mpicc -o test test.c
```

Заметим, что для нашей игрушечной программы такая команда компиляции вполне подходит, но для настоящей программы следовало бы написать что-нибудь вроде:

```
mpicc -o test -O3 -ffast-math -march=i686 test.c -lm
```

После компиляции программу можно запустить на 1-м процессоре:

```
mpirun -np 1 test
```

В результате получится выдача:

```
Total processors number: 1
Starting process N 0 on oryx1.bog.msu.ru
Total sum: 1, max: 1, time: 0.04
```

Эту же программу можно запустить и на 4-х процессорах:

```
mpirun -np 4 test
```

Тогда получится выдача:

```
Total processors number: 4
Starting process N 0 on oryx1.bog.msu.ru
Starting process N 2 on oryx3.bog.msu.ru
Starting process N 3 on oryx4.bog.msu.ru
Starting process N 1 on oryx2.bog.msu.ru
Total sum: 10, max: 4, time: 0.03
```

Ну а теперь дадим необходимые объяснения.

Как видно из приведенного примера, прежде всего в обычный перечень директив `#include` необходимо добавить `#include<mpi.h>`.

Инициализация протокола MPI производится функцией

```
MPI_Init(&npar, &par);
```

Начиная с вызова этой функции, можно использовать другие команды MPI. Аргументы функции суть адреса аргументов функции `main`.³ Завершение протокола MPI производится функцией

```
MPI_Finalize();
```

³Полезно напомнить, что аргументы функции `main` описывают параметры командной строки при запуске программы: «праг» — число параметров командной строки, «par [0]» — строка, содержащая имя исполняемого файла, «par [1]» — строка, содержащая первый параметр в командной строке, «par [2]» — строка, содержащая второй параметр, «par [праг-1]» — строка, содержащая последний параметр.

Начиная с момента вызова этой функции, использовать команды MPI уже нельзя.

Первое, что должна сделать распараллеленная программа, работающая на данном процессоре, это определить, сколько всего задействовано процессоров. Количество процессоров определяется с помощью функции

```
MPI_Comm_size( MPI_COMM_WORLD, &totnum);
```

После вызова этой функции целая переменная *totnum* будет равна полному числу процессоров. Первый аргумент функции введен из следующих соображений: в MPI есть возможность определять группы процессоров, являющихся подмножествами множества всех задействованных процессоров. Мы в дальнейшем будем игнорировать эту возможность, так что единственная группа, с которой мы будем иметь дело — это группа всех процессоров, которая называется MPI_COMM_WORLD. Так что во всех функциях, где требуется имя группы, мы будем использовать MPI_COMM_WORLD.

Кроме того, надо определить, каков индивидуальный номер данного процессора. Это делается с помощью функции

```
MPI_Comm_rank( MPI_COMM_WORLD, &mynum);
```

После вызова этой функции целая переменная *mynum* будет равна номеру данного процессора, а именно: на одном из процессоров она будет равна 0, на другом — 1, на третьем — 2, ..., на последнем — (*totnum* - 1).

Наконец, можно определить имя данного процессора (для MPI на базе локальной сети это попросту *host name*). Это делается с помощью функции

```
MPI_Get_processor_name(name, &len);
```

После вызова этой функции в строковой переменной *name*, которую надлежит определять как

```
char name[MPI_MAX_PROCESSOR_NAME];
```

будет содержаться имя процессора (для компьютерного кластера это будет *host name* в формате *mmm.hhh.ru*), а целая переменная *len* будет равна длине имени данного процессора.

Нередко выяснить имя машины действительно необходимо. Например, если задействованные компьютеры имеют разные производительности и разные объемы памяти, то необходимо выделить каждому процессору долю задачи, соответствующую его параметрам. Если поделить задачу поровну, то скорость счета будет определяться скоростью самого медленного процессора, а остальные будут его ждать, т. е. простаивать.

Разумеется, должны существовать команды, реализующие обмен информацией между процессорами. Данный процессор (пусть его номер равен

numproc0) может послать numdat штук целых чисел, размещенных в массиве armdat (число numdat может быть меньше декларированной длины массива), процессору с номером numproc1, если использовать функцию

```
MPI_Send(armdat, numdat, MPI_INT, numproc1,  
         index, MPI_COMM_WORLD);
```

Тип посылаемых данных указывается с помощью аргумента MPI_INT. При необходимости послать содержимое одной целой переменной dat имя массива armdat⁴ надо заменить на адрес переменной &dat, а число numdat положить равным 1. Совершенно аналогично можно послать массив double-чисел, достаточно только заменить MPI_INT на MPI_DOUBLE. Аргумент index (это переменная типа int) есть «метка» сообщения. Метка, придаваемая сообщению, нередко действительно необходима: дело в том, что при посылке двух сообщений подряд не гарантируется, что они придут в том же порядке, в котором были посланы. Чтобы можно было различать разные сообщения, они снабжаются метками.

Чтобы процессор с номером numproc1 принял это сообщение, необходимо использовать функцию

```
MPI_Recv(armdat, numdat, MPI_INT, numproc0,  
         index, MPI_COMM_WORLD, &stat);
```

В принципе нет необходимости принимать данные в тот же самый массив armdat, в котором они находились на процессоре с номером numproc0. Обязаны совпадать только тип данных, их количество и метка сообщения index. Дополнительный аргумент &stat содержит диагностику ошибок, которые могут произойти при приеме сообщения. Опыт показывает, что пользы от этой диагностики немного, поэтому мы не будем ее описывать.

Приведенные функции являются «блокирующими» функциями приема и передачи. Это означает, что программа ожидает, пока сообщение не будет отослано или не будет принято, и только после этого выполнение программы продолжится. Следует иметь в виду, что существуют и другие типы функций приема и передачи («неблокирующие», «синхронные» и т.д.), однако опыт показывает, что в практической работе можно обойтись и приведенными здесь функциями.

В нашем примере программа выполняет следующие действия: процессор номер 0 присваивает переменной dat значение 1 и посылает его процессору номер 1. Остальные процессоры в это время простаивают, поскольку ожидают своих сообщений. Процессор номер 1 получает сообщение от процессора номер 0 (одно целое число, равное 1) и помещает его в «свою»

⁴Полезно напомнить, что имя массива есть адрес первого элемента массива.

переменную *dat*. Затем увеличивает переменную *dat* на 1 и посылает получившееся число 2 процессору номер 2. И так далее, вплоть до последнего процессора номер (*totnum* - 1), который получит от предыдущего процессора число (*totnum* - 1), увеличит его до *totnum*, но уже никуда не пошлет. Такова довольно бессмысленная цепочка обмена сообщениями в этой программе.

Иногда возникает необходимость послать одно и то же сообщение от одного из процессоров всем остальным. Разумеется, можно было бы заставить этот один процессор рассылать сообщения всем остальным с помощью многократного вызова функции `MPI_Send`, а остальные — принимать эти сообщения с помощью `MPI_Recv`. Однако гораздо удобнее воспользоваться функцией

```
MPI_Bcast(arrdat, numdat, MPI_INT, numproc0,
          MPI_COMM_WORLD);
```

Смысл аргументов *arrdat*, *numdat* и `MPI_INT` очевиден, а аргумент *numproc0* указывает, что именно процессор с номером *numproc0* рассылает данные всем остальным процессорам. После вызова функции первые *numdat* элементов массива *arrdat* становятся идентичными на всех процессорах — они становятся такими же, как на процессоре с номером *numproc0*.

Ясно, что в разрезанной на кусочки задаче, скорее всего, возникнет необходимость так или иначе объединить данные, полученные для каждого отдельного кусочка: или найти полную энергию, которая есть сумма энергий каждого из кусочков задачи, или найти максимум погрешности, которая есть максимум среди максимальных погрешностей для каждого кусочка, и т. п.

Очевидно, что эти операции легко реализовать с помощью функций `MPI_Send` и `MPI_Recv`. Однако, удобнее воспользоваться стандартной функцией

```
MPI_Reduce(arrdat, totdat, numdat, MPI_INT,
           MPI_SUM, numproc0, MPI_COMM_WORLD);
```

В результате вызова этой функции произойдет следующее: все элементы массива *totdat* на процессоре с номером *numproc0* с индексами $k = 0, 1, \dots, (\text{numdat} - 1)$ примут значения

$$\text{totdat}[k] = \sum_{n=0}^{N-1} \text{arrdat}_n[k],$$

где $N = \text{totnum}$ — полное число процессоров, а $\text{arrdat}_n[k]$ — значение *k*-го элемента массива *arrdat* на *n*-ом процессоре. Оба массива должны

иметь тип `int`. Заметим, что элементы массива `totdat` меняются только на процессоре с номером `numproc0`, на остальных процессорах они остаются неизменными.

Помимо очевидных аргументов `arrdat`, `totdat`, `numdat`, `MPI_INT`, здесь есть еще и аргумент `MPI_SUM`, указывающий на тип операции, которую надлежит проделать над данными. Кроме суммирования (`MPI_SUM`), существует `MPI_PROD` — умножение, `MPI_MAX` — максимум, `MPI_MIN` — минимум.

В приведенном нами примере результатом выполнения первого вызова `MPI_Reduce` (с аргументом `MPI_SUM`) будет

$$\sum_{m=1}^N m = N(N+1)/2.$$

Соответственно, результатом второго вызова `MPI_Reduce` (с аргументом `MPI_MAX`) будет

$$\max_{m=1 \dots N} m = N.$$

Если нам необходимо разослать результат вызова `MPI_Reduce` по всем процессорам, то можно, конечно, воспользоваться функцией `MPI_Bcast`. Удобнее, однако, использовать функцию

```
MPI_Allreduce(arrdat, totdat, numdat, MPI_INT,
               MPI_SUM, MPI_COMM_WORLD);
```

которая отличается от функции `MPI_Reduce` только тем, что помещает результаты вычислений в массив `totdat` не на одном процессоре, а сразу на всех задействованных процессорах.

Упомянем и некоторые другие функции.

Функция

```
MPI_Barrier(MPI_COMM_WORLD);
```

останавливает выполнение программы на данном процессоре до тех пор, пока остальные процессоры также не доберутся до этой функции. Тем самым реализуется синхронизация в выполнении программы. Эта функция, безусловно, полезна при отладке, но в отлаженной программе, где используются блокирующие операторы `MPI_Send` и `MPI_Recv`, она не особенно полезна.

Функция

```
MPI_Abort(MPI_COMM_WORLD, number);
```


прерывает выполнение программы и возвращает системе целое число `number` в качестве кода завершения (кода ошибки).⁵ Опять-таки, это довольно полезная при отладке функция, но завершать таким образом работу отлаженной программы не рекомендуется (под Linux это вызывает огромное количество системных ошибок типа `broken pipe...`).

Существует довольно полезная функция

```
MPI_wtime();
```

которая возвращает `double`-число, равное времени (в секундах), которое миновало с момента инициализации протокола MPI.

Необходимо подчеркнуть, что мы упомянули лишь небольшую часть из более чем 300 функций и констант, определенных в MPI, однако, даже этих элементарных сведений вполне достаточно для написания реальных распараллеленных программ.

3. Задача Коши для обыкновенных дифференциальных уравнений

Постановка задачи Коши для системы обыкновенных дифференциальных уравнений (ОДУ) вполне тривиальна. Прежде всего очевидно, что любая система ОДУ произвольного порядка может быть переписана в виде системы ОДУ первого порядка:

$$\frac{d}{dt}x_i(t) = f_i(x_1(t), \dots, x_n(t)), \quad i = 1, \dots, n.$$

Переменную t мы будем в дальнейшем называть «временем», кроме того, мы часто будем использовать векторные обозначения: $\frac{d}{dt}\vec{x} = \vec{f}(\vec{x})$. Заметим, что нет необходимости отдельно рассматривать случай, когда правая часть в системе ОДУ явно зависит от времени, поскольку при этом достаточно определить «лишнюю» переменную $x_{n+1}(t) \equiv t$, которая удовлетворяет уравнению $\frac{d}{dt}x_{n+1}(t) = 1$. Задача Коши состоит в том, что при известных начальных данных $\vec{x}(t = t_1)$ необходимо проследить эволюцию системы вплоть до некоторого момента времени t_2 и найти $\vec{x}(t_2)$.

Решение дифференциальных уравнений (наряду с обработкой данных) почему-то считается основным разделом курса численных методов, так что большинство руководств уделяет основное внимание именно этому вопросу. Мы, напротив, будем довольно лаконичны.

⁵Это то же самое, что в функции `main` написать `return number;`.

Мы не станем рассматривать методы, которые применимы только для некоторого класса ОДУ, в частности, мы не будем обсуждать метод Нумерова. Этот метод действительно довольно эффективен для уравнений второго порядка, но он не настолько превосходит по эффективности универсальные методы, чтобы однозначно рекомендовать его применение. Вообще, при практической работе зачастую целесообразнее использовать готовую отлаженную программу, использующую общий метод, чем заново создавать новую (пусть и чуть более эффективную) программу для каждого отдельного случая.

3.1. Методы Рунге-Кутты

Группа методов, которые мы будем называть методами Рунге-Кутты, основана на очень простой идее: мы хотим сдвинуться по времени на достаточно маленькую величину h (если правая часть ОДУ порядка единицы, то $h = 10^{-2} \dots 10^{-4}$, в зависимости от заказанной точности). Мы используем разложение в ряд Тейлора

$$\vec{x}(t+h) = \vec{x}(t) + \dot{\vec{x}}(t) \frac{h^1}{1!} + \ddot{\vec{x}}(t) \frac{h^2}{2!} + \dots,$$

при этом

$$\dot{\vec{x}}(t) = \vec{f}(\vec{x}(t)),$$

$$\ddot{x}_i(t) = \partial_{x_j} f_i(\vec{x}(t)) \dot{x}_j(t) = \partial_{x_j} f_i(\vec{x}(t)) f_j(x(t))$$

и т.д.

Вычисление производных $\frac{d^n}{dt^n} \vec{x}(t)$ осуществляется посредством «прошупывания» траектории на отрезке $[t, t+h]$, т.е. вычисления правой части ОДУ в промежуточных точках. Разные схемы типа Рунге-Кутты отличаются друг от друга как порядком (т.е. количеством промежуточных точек и, следовательно, количеством вычисленных производных), так и положением промежуточных точек.

3.1.1. Разнообразные n -точечные схемы

Простейшая схема типа Рунге-Кутты — это схема второго порядка. Мы вычисляем правую часть ОДУ в двух точках:

$$\vec{f}_1 = \vec{f}(\vec{x}(t) + \delta \vec{x}_1), \quad \delta \vec{x}_1 = 0,$$

$$\vec{f}_2 = \vec{f}(\vec{x}(t) + \delta \vec{x}_2), \quad \delta \vec{x}_2 = h a_2 \vec{f}_1,$$

где a_2 — произвольная константа. При этом

$$(f_1)_i = \dot{x}_i(t),$$

$$(f_2)_i \approx (f_1)_i + a_2 h \partial_{x_j} f_i(\vec{x}(t)) f_j(x(t)) = \dot{x}_i(t) + a_2 h \ddot{x}_i(t).$$

Ищем смещение по траектории в виде

$$\delta \vec{x} = x(t+h) - x(t) \approx w_1 h \vec{f}_1 + w_2 h \vec{f}_2.$$

Следовательно,

$$\ddot{x}(t) \frac{h^1}{1!} + \ddot{x}(t) \frac{h^2}{2!} = w_1 h \dot{x}_i(t) + w_2 h [\dot{x}_i(t) + a_2 h \ddot{x}_i(t)],$$

откуда

$$w_2 = \frac{1}{2a_2} \quad w_1 = 1 - w_2$$

с точностью до слагаемых $O(h^2)$ при произвольном a_2 .

Все семейство схем второго порядка с разнообразными a_2 имеет одинаковый порядок точности. Однако для каждой данной системы ОДУ фактическая погрешность будет (хотя и не слишком сильно) зависеть от a_2 . Причем для разных задач мы будем получать разные зависимости от a_2 . Из эстетических (симметричных) соображений можно выбрать $a_2 = 1/2$, что соответствует $\delta \vec{x} = h \vec{f}_2$. Можно также выбрать $a_2 = 1$, что дает $\delta \vec{x} = (h/2) \vec{f}_1 + (h/2) \vec{f}_2$. К сожалению, гипотеза о том, что симметричные схемы могут быть на один порядок точнее несимметричных схем (кажется, что коэффициенты при нечетных степенях h могли бы обращаться в ноль, как это имеет место в квадратурных формулах — см. часть I, раздел «Численное интегрирование»), опровергается непосредственной проверкой.

Разумеется, пользоваться схемами второго порядка не следует.

Совершенно аналогичным образом может быть построена схема третьего порядка. Правая часть ОДУ вычисляется в трех точках:

$$\vec{f}_k = \vec{f}(\vec{x}(t) + \delta \vec{x}_k),$$

где

$$\delta \vec{x}_1 = 0, \quad \delta \vec{x}_2 = h a_2 \vec{f}_1, \quad \delta \vec{x}_3 = h(a_3 - b_3) \vec{f}_1 + h b_3 \vec{f}_2.$$

Смещение по траектории ищем в виде

$$\delta \vec{x} = w_1 \vec{f}_1 + w_2 \vec{f}_2 + w_3 \vec{f}_3.$$

Сравнивая ряд Тейлора с полученным выражением, получаем либо

$$w_2 = \frac{2 - 3a_3}{6a_2(a_2 - a_3)}, \quad w_3 = \frac{2 - 3a_2}{6a_3(a_3 - a_2)}, \quad b_3 = \frac{a_3(a_2 - a_3)}{a_2(3a_2 - 2)}$$

при произвольных a_2, a_3 ; либо $a_2 = 2/3, a_3 = 2/3, w_2 = 3/4 - 1/(4b_3), w_3 = 1/(4b_3)$ при произвольном b_3 ; либо $a_2 = 2/3, a_3 = 0, w_2 = 3/4, w_3 = 1/(4b_3)$ при произвольном b_3 . Во всех случаях $w_1 = 1 - w_2 - w_3$. Эти наборы параметров обеспечивают точность $O(h^3)$.

Для каждой конкретной системы ОДУ фактическая погрешность зависит от выбранных параметров, но не слишком сильно. Кроме того, для разных задач эта зависимость оказывается разной.

Разумеется, пользоваться схемой третьего порядка не следует.

Точно так же строится схема четвертого порядка:

$$\vec{f}_k = \vec{f}(\vec{x}(t) + \delta \vec{x}_k),$$

где

$$\delta \vec{x}_1 = 0, \quad \delta \vec{x}_2 = ha_2 \vec{f}_1, \quad \delta \vec{x}_3 = h(a_3 - b_3) \vec{f}_1 + hb_3 \vec{f}_2,$$

$$\delta \vec{x}_4 = h(a_4 - b_4 - c_4) \vec{f}_1 + hb_4 \vec{f}_2 + hc_4 \vec{f}_3.$$

Смещение по траектории ищем в виде

$$\delta \vec{x} = w_1 \vec{f}_1 + w_2 \vec{f}_2 + w_3 \vec{f}_3 + w_4 \vec{f}_4.$$

Сравнивая ряд Тейлора с полученным выражением, легко получить решение для параметров $b_3, b_4, c_4, w_1, w_2, w_3, w_4$ при произвольных a_2, a_3, a_4 . Решение это не представляет особого интереса и слишком громоздко, чтобы его здесь приводить. Мы ограничимся частным случаем, удовлетворяющим условию симметрии:

$$a_2 = 1/2 - m, \quad a_3 = 1/2 + m, \quad a_4 = 1$$

при произвольном m . Тогда

$$b_3 = \frac{1 + 2m}{2 - 4m}, \quad b_4 = \frac{2m(1 + 2m)}{(1 - 2m)(1 - 12m^2)},$$

$$c_4 = \frac{1 - 2m}{1 - 12m^2},$$

$$w_1 = w_4 = \frac{1 - 12m^2}{6 - 24m^2}, \quad w_2 = w_3 = \frac{1}{3 - 12m^2}.$$

«Классическая» схема, которую обыкновенно и называют методом Рунге-Кутты, получается при $m = 0$:

$$\begin{aligned}\vec{f}_1 &= \vec{f}(\vec{x}(t)), & \vec{f}_2 &= \vec{f}(\vec{x}(t) + \frac{h}{2}\vec{f}_1), \\ \vec{f}_3 &= \vec{f}(\vec{x}(t) + \frac{h}{2}\vec{f}_2), & \vec{f}_4 &= \vec{f}(\vec{x}(t) + h\vec{f}_3), \\ \delta\vec{x} &= \frac{h}{6}\vec{f}_1 + \frac{h}{3}\vec{f}_2 + \frac{h}{3}\vec{f}_3 + \frac{h}{6}\vec{f}_4.\end{aligned}$$

При $m = 1/6$ получим:

$$\begin{aligned}a_2 &= 1/3, & a_3 &= 2/3, & a_4 &= 1, & b_3 &= 1, & b_4 &= -1, \\ c_4 &= 1, & w_1 &= w_4 = 1/8, & w_2 &= w_3 = 3/8.\end{aligned}$$

При $m = 1/14$ получим:

$$\begin{aligned}a_2 &= 3/7, & a_3 &= 4/7, & a_4 &= 1, & b_3 &= 2/3, & b_4 &= -14/69, \\ c_4 &= 21/23, & w_1 &= w_4 = 23/144, & w_2 &= w_3 = 49/144.\end{aligned}$$

И так далее.

Легко проверить, что ни по точности, ни по скорости счета эти схемы не слишком отличаются друг от друга (для любой реальной задачи экономия трех умножений в «классической» схеме незаметна на фоне вычисления правой части ОДУ). Единственным достоинством «классической» схемы является простота соответствующих выражений.

Следует ли применять схемы четвертого порядка? Обыкновенно утверждается, что схемы четвертого порядка являются оптимальными по производительности, т. е. при заданной точности они соответствуют минимальному количеству операций, затрачиваемых на единичный сдвиг $t \rightarrow t + 1$. По нашему мнению, схемы пятого порядка все же в среднем более производительны.

Кроме того, как правило, утверждается, что ни в коем случае не следует применять схемы Рунге-Кутты без контроля точности и без адаптивного изменения шага. В общем случае это безусловно правильное утверждение.

Однако если речь идет о задаче, в которой общий характер поведения правой части ОДУ известен, и эта правая часть определенно не будет испытывать резких изменений вдоль траектории, то вполне можно обойтись без адаптивного изменения шага. Точность легко оценить по небольшому участку траектории, пройдя его один раз с выбранным шагом, а второй раз — с шагом, уменьшенным в 1.5 или 2 раза, и сравнив ответы.

Другая ситуация, при которой необязательно менять шаг, возникает при неоднократном решении некоторой системы ОДУ на одном и том же отрезке времени, но со слегка отличающимися параметрами (слегка отличающиеся начальные условия или слегка отличающиеся числовые параметры, входящие в правую часть ОДУ). Типичный пример — решение краевой задачи для ОДУ методом стрельбы (см. раздел «Краевая задача для ОДУ»). В этом случае можно один раз (для данного набора параметров) найти постоянный шаг, обеспечивающий необходимую точность для всей траектории. Опять-таки, точность легко оценить, измельчив шаг в 1.5 или 2 раза, и сравнивая ответы. Найденный таким образом шаг используется для остальных наборов параметров уже без проверки.

В этих (и только в этих) случаях действительно можно применять «классическую» схему Рунге-Кутты четвертого порядка без контроля точности и без адаптивного изменения шага.

3.1.2. Адаптивное изменение шага

В общем случае совершенно необходимо применять локальный контроль точности и менять шаг h . Следует понимать, что адаптивное изменение шага делается не только ради увеличения производительности, оно необходимо и для достижения заказанной точности. Дело в том, что уменьшение шага приводит к улучшению точности только до определенного предела. Начиная с некоторого шага, точность начинает падать (для типичных ОДУ с правой частью порядка единицы этот предельный шаг порядка $h = 10^{-4} \dots 10^{-6}$). Это происходит потому, что мы делаем слишком много шагов на отрезке $[t_1, t_2]$, и, следовательно, накапливаем большую ошибку округления. Поэтому для достижения необходимой «глобальной» точности (точности ответа $\vec{x}(t_2)$, достигнутой на правом конце временного интервала) совершенно необходимо на разных участках траектории обеспечивать постоянную «локальную» точность (точность на одном шаге) — не хуже, но и не лучше заказанной. Достигается это за счет изменения шага: «простые» отрезки траектории (небольшая и плавно меняющаяся правая часть) следует проходить с достаточно большим шагом (минимизируя ошибки округления), а на «сложных» отрезках (большая и резко меняющаяся правая часть) шаг нужно уменьшать для достижения нужной локальной точности.

Простейший, хотя и крайне неэффективный способ контроля точности (и заодно увеличения порядка схемы на единицу) заключается в следующем: с помощью «классической» схемы четвертого порядка сделаем из точки $\vec{x}(t)$ два шага подряд на величину h (суммарный сдвиг по времени

будет $2h$), получим точку $\vec{x}'(t+2h) = \vec{x}(t) + \delta\vec{x}'$. Затем сделаем из исходной точки $\vec{x}(t)$ один шаг, но уже на $2h$, получим точку $\vec{x}''(t+2h) = \vec{x}(t) + \delta\vec{x}''$. Для схем четвертого порядка погрешность пропорциональна пятой степени шага, т.е. равна $\vec{A}h^5$, причем на интервале $[t, t+2h]$ мы будем считать $\vec{A}$ приблизительно постоянным. Следовательно, истинное значение $\vec{x}(t+2h)$ можно оценить либо как $\vec{x}(t+2h) \approx \vec{x}'(t+2h) + 2\vec{A}h^5$, либо как $\vec{x}(t+2h) \approx \vec{x}''(t+2h) + \vec{A}(2h)^5$. Поэтому ошибку можно оценить как $\vec{E} = \delta\vec{x}' - \delta\vec{x}''$, а «истинное» значение $\vec{x}(t+2h)$ как $\vec{x}(t+2h) = \vec{x}(t) + \delta\vec{x}' + (\delta\vec{x}'' - \delta\vec{x}')/15$ с точностью до $O(h^5)$. Заметим, что ошибка $\vec{E}$ — это ошибка ответа $\vec{x}'(t+2h)$, а вовсе не ошибка окончательного (поправленного) ответа $\vec{x}(t+2h)$.

Довольно ясно, что такой рецепт крайне неэффективен. Здесь приходится 11 раз вычислять правую часть ОДУ (три шага схемы Рунге-Кутты соответствуют 12 вычислениям, но одна (начальная) точка общая). Достигаем же мы всего-навсего ответа $\vec{x}(t+2h)$ пятого порядка точности и еще одного дополнительного ответа $\vec{x}'(t+2h)$ четвертого порядка точности, что позволяет нам оценить погрешность $\vec{E}$. Очевидно, что этих результатов можно достичь, вычисляя правую часть в гораздо меньшем количестве точек. Для получения ответа пятого порядка точности $\vec{x}(t+h) = \vec{x}(t) + \delta\vec{x}$ достаточно пяти точек, а чтобы получить еще и дополнительный ответ $\vec{x}'(t+h) = \vec{x}(t) + \delta\vec{x}'$ четвертого порядка точности, надо добавить шестую точку:

$$\vec{f}_i = \vec{f}(\vec{x} + \delta\vec{x}_i),$$

где

$$\delta\vec{x}_1 = 0,$$

$$\delta\vec{x}_2 = h a_2 \vec{f}_1,$$

$$\delta\vec{x}_3 = h(a_3 - b_3)\vec{f}_1 + h b_3 \vec{f}_2,$$

$$\delta\vec{x}_4 = h(a_4 - b_4 - c_4)\vec{f}_1 + h b_4 \vec{f}_2 + h c_4 \vec{f}_3,$$

$$\delta\vec{x}_5 = h(a_5 - b_5 - c_5 - d_5)\vec{f}_1 + h b_5 \vec{f}_2 + h c_5 \vec{f}_3 + h d_5 \vec{f}_4,$$

$$\delta\vec{x}_6 = h(a_6 - b_6 - c_6 - d_6 - e_6)\vec{f}_1 + h b_6 \vec{f}_2 + h c_6 \vec{f}_3 + h d_6 \vec{f}_4 + h e_6 \vec{f}_5.$$

Смещения $\delta\vec{x}$ и $\delta\vec{x}'$ мы ищем в виде

$$\delta\vec{x} = h w_1 \vec{f}_1 + h w_2 \vec{f}_2 + h w_3 \vec{f}_3 + h w_4 \vec{f}_4 + h w_5 \vec{f}_5 + h w_6 \vec{f}_6,$$

$$\delta\vec{x}' = h v_1 \vec{f}_1 + h v_2 \vec{f}_2 + h v_3 \vec{f}_3 + h v_4 \vec{f}_4 + h v_5 \vec{f}_5 + h v_6 \vec{f}_6,$$

причем

$$w_1 = 1 - w_2 - w_3 - w_4 - w_5 - w_6,$$

$$v_1 = 1 - v_2 - v_3 - v_4 - v_5 - v_6.$$

Сравнивая ряд Тейлора с полученным выражением, получаем недоопределенную систему нелинейных уравнений на параметры.

Одним из решений является набор параметров, соответствующий «классической» схеме пятого порядка:

a_2	1/5	d_6	44275/110592
a_3	3/10	e_6	253/4096
a_4	3/5	w_2	0
a_5	1	w_3	250/621
a_6	7/8	w_4	125/594
b_3	9/40	w_5	0
b_4	-9/10	w_6	512/1771
b_5	5/2	v_2	0
b_6	175/512	v_3	18575/48384
c_4	6/5	v_4	13525/55296
c_5	-70/27	v_5	277/14336
c_6	575/13824	v_6	1/4
d_5	35/27		

Именно это решение, как правило, и используется.

В действительности для произвольного набора шести точек, которые определяются коэффициентами $a_2 \dots a_6$, можно численно найти соответствующее решение. Более того, для набора $a_2 = 1/5$, $a_3 = 2/5$, $a_4 = 3/5$, $a_5 = 4/5$, $a_6 = 1$ система уравнений испытывает дополнительное вырождение, так что для этого набора точек есть целое семейство решений.

Однако, в отличие от схем четвертого порядка, все эти решения существенно отличаются друг от друга по своим свойствам.

Прежде всего, они отличаются друг от друга величиной коэффициентов при остаточных членах шестого порядка. Для разных четырехточечных схем четвертого порядка коэффициенты при остаточных членах пятого порядка практически одинаковы. Точно так же для разных пятиточечных схем пятого порядка коэффициенты при остаточных членах шестого порядка практически одинаковы. Но в данном случае мы строим разные шеститочечные схемы пятого порядка, так что коэффициенты при остаточных членах шестого

порядка могут существенно отличаться по величине. Существуют решения, которые на порядок хуже «классической» схемы (т. е. коэффициенты при остаточных членах в 10 раз больше), но существуют и решения, которые на порядок лучше.

Осуществить сравнение разных схем и дать универсальную рекомендацию о выборе лучшей схемы не так-то просто, поскольку для разных систем ОДУ получаются разные результаты. Например, для почти линейных задач (многомерный осциллятор со слабой нелинейностью) вторая, третья и прочие высшие производные в правой части ОДУ практически равны нулю. Так что существенен только один из коэффициентов в остаточном члене. Легко соорудить схему, которая для таких почти линейных задач будет на два порядка лучше всех остальных схем. Однако эта схема, почти идеальная для квазилинейных задач, потерпит полную катастрофу в существенно нелинейном случае.

Тем не менее существуют альтернативные схемы, которые заметно лучше «классической» схемы пятого порядка, например:

a_2	0.737497447523991	d_6	-0.0227559457060132
a_3	0.514578737659973	e_6	0.221179093947989
a_4	0.218659346963748	w_2	0.0104976125436066
a_5	0.584781031573568	w_3	0.0139546219366905
a_6	0.906489681300872	w_4	0.326178536183172
b_3	0.745196018656341	w_5	0.354711363752133
b_4	-0.0151561539013055	w_6	0.227610779403418
b_5	0.0458409558302733	u_2	-0.526096724604447
b_6	0.231885467774540	u_3	-0.318776777304289
c_4	0.0677459231515263	u_4	-0.0998859236674947
c_5	-0.0774644495257406	u_5	0.770101117841966
c_6	0.245178009225538	u_6	0.136274257156700
d_5	0.740992484018385		

Здесь $u_i \equiv w_i - v_i$. Знание u_i позволяет сразу вычислить ошибку $\vec{E} = \delta\vec{x} - \delta\vec{x}'$:

$$\vec{E} = hu_1\vec{f}_1 + hu_2\vec{f}_2 + hu_3\vec{f}_3 + hu_4\vec{f}_4 + hu_5\vec{f}_5 + hu_6\vec{f}_6.$$

Эта схема лучше «классической» схемы пятого порядка, так что именно ее мы и рекомендуем.

Итак, если Вы собираетесь контролировать точность и менять шаг, то следует использовать приведенную схему пятого порядка. Применение простейшего рецепта с двумя шагами по h и одним шагом на $2h$ затормозит выкладки почти в два раза.

Теперь обсудим вопрос о том, как именно следует менять шаг. После одного шага, выполненного с помощью схемы пятого порядка, мы располагаем значением $\vec{x}(t+h)$ и ошибкой $\vec{E}$. Следует понимать, что:

- во-первых, ошибка $\vec{E}$ — это вовсе не ошибка в $\vec{x}(t+h)$, это ошибка в $\vec{x}'(t+h)$, т. е. в величине, полученной с помощью схемы четвертого порядка;
- во-вторых, ошибка $\vec{E}$ суть вектор, т. е. мы располагаем ошибками во всех переменных x_i' по отдельности. Нам надо превратить вектор $\vec{E}$ в одно число — в величину погрешности ϵ , характеризующую ошибку, которую мы получили, сделав шаг $t \rightarrow t+h$. Это позволит выбрать величину следующего шага по времени h_0 , руководствуясь заказанной точностью решения ОДУ ϵ_0 . К сожалению, универсального способа превратить вектор $\vec{E}$ в одно число ϵ не существует.

— Существуют задачи, в которых все переменные x_i меняются в некотором ограниченном интервале, причем их абсолютные величины приблизительно одинаковы. В этом случае довольно естественно использовать абсолютную погрешность: $\epsilon = \max_i |E_i|$, либо $\epsilon = \sqrt{\sum_i E_i^2}$.

— Существуют задачи, в которых масштабы переменных существенно разные, так что приходится использовать относительные погрешности: $\epsilon = \max_i |E_i/x_i|$, либо $\epsilon = \sqrt{\sum_i (E_i/x_i)^2}$.

Иногда более правильно использовать относительные погрешности, но не по отношению к самим величинам x_i , а по отношению к их приращениям: $\epsilon = \max_i |E_i/[x_i(t+h) - x_i(t)]|$, либо $\epsilon = \sqrt{\sum_i (E_i/[x_i(t+h) - x_i(t)])^2}$.

При использовании относительных погрешностей надо быть очень осторожным. Знаменатели могут обращаться в ноль — в этом случае мы катастрофически завысим требования. И наоборот, знаменатели могут резко возрастать — в этом случае требования будут существенно занижены.

— Наконец, существуют задачи, в которых на каждую из переменных накладывается свое индивидуальное требование — в этом случае приходится полагать $\epsilon = \max_i |E_i/\delta_i|$, где каждое δ_i выбирается, исходя из требований к точности каждой данной переменной.

После того, как мы вычислили ϵ , выбор величины следующего шага h_0 превращается в тривиальную задачу. Раз шаг h привел к погрешности ϵ , которая пропорциональна пятой степени h ($\epsilon = Ah^5$), то для шага h_0 , который приведет к заказанной точности ϵ_0 , мы получаем: $\epsilon_0/\epsilon = (h_0/h)^5$, откуда $h_0 = h(\epsilon_0/\epsilon)^{1/5}$. На практике используют несколько иной рецепт.

Алгоритм:

Решая задачу Коши для ОДУ на отрезке $[t_1, t_2]$, повторяем следующие действия, пока t не достигнет t_2 :

Выполняем один шаг по времени $\vec{x}(t) \rightarrow \vec{x}(t+h)$ с помощью схемы пятого порядка, в результате получаем $\vec{x}(t+h)$ и $\vec{E}$.

Вычисляем ϵ .

Если $\epsilon < \epsilon_0$,

вычисляем $h_0 = Dh(\epsilon_0/\epsilon)^{1/5}$, где $D = 0.8 \dots 0.9$. Проверяем, что $t+h+h_0 < t_2$, в противном случае укорачиваем новый шаг h_0 до $h_0 = t_2 - t - h$. Принимаем $\vec{x}(t+h)$ как новую точку на траектории, т. е. переприсваиваем $\vec{x}(t+h) \rightarrow \vec{x}(t)$.

В противном случае

вычисляем $h_0 = Dh(\epsilon_0/\epsilon)^{1/4}$, где $D = 0.8 \dots 0.9$. Отвергаем $\vec{x}(t+h)$ и делаем новую попытку с меньшим шагом, начиная со старой точки $\vec{x}(t)$.

Переприсваиваем $h = h_0$.

Здесь D — страховочный множитель, который предотвращает крайне нежелательную необходимость отвергать очередную точку на траектории. Наличие D позволяет надеяться, что если вдруг в правой части ОДУ произойдут изменения, то ϵ возрастет по сравнению с «запланированным» значением, но все же не превысит ϵ_0 . Действительно, «запланированная» погрешность составляет $D^5\epsilon_0 = (0.3 \dots 0.6)\epsilon_0$. В идеальном случае ϵ меньше ϵ_0 на величину D^5 , а h плавно меняется в зависимости от поведения правой части ОДУ.

Степень «1/4» вместо «1/5» при уменьшении шага следует воспринимать просто как рецепт, оказавшийся эффективным на практике. Существует квазиобъяснение, что при этом мы заботимся о глобальной точности: локальная точность (точность данного шага) пропорциональна h^5 . При этом количество шагов пропорционально $1/h$, так что глобальная точность (точность на конце временного интервала) пропорциональна h^4 . Если пользоваться этим объяснением, то не очень понятно, отчего же в другом ме-

сте стоит $\langle 1/5 \rangle$. В действительности объяснение такое: всегда лучше чуть занижить шаг, чем его завысить, так что при $\epsilon < \epsilon_0$ мы берем меньшую из величин $-Dh(\epsilon_0/\epsilon)^{1/5}$, при $\epsilon > \epsilon_0$ мы также берем меньшую из величин $-Dh(\epsilon_0/\epsilon)^{1/4}$.

Итак, методы Рунге-Кутты пятого порядка с контролем точности и адаптивным изменением шага вполне можно применять, в особенности для задач с резко меняющейся правой частью. Для задач с плавно меняющейся правой частью они тоже приемлемы, но заметно проигрывают по точности и очень сильно проигрывают по производительности интерполяционным методам.

3.2. Интерполяционные методы

Основная идея интерполяционных методов при решении задачи Коши для ОДУ очень проста. Она в точности совпадает с идеей, заложенной в методе Ромберга (см. часть I, раздел «Численное интегрирование»).

Давайте пройдем один и тот же отрезок траектории $t \rightarrow t + H$ сначала с шагом $h_1 = H/N_1$, затем с меньшим шагом $h_2 = H/N_2$, затем с еще меньшим шагом $h_3 = H/N_3$ и т.д. При этом мы получим точки $\vec{x}^{(1)}(t+H)$, $\vec{x}^{(2)}(t+H)$, $\vec{x}^{(3)}(t+H)$ и т.д. В отличие от метода Ромберга, здесь при уменьшении шага невозможно использовать результаты, полученные с более крупным шагом, поэтому не обязательно уменьшать шаг в два раза — шаг можно уменьшать как угодно. Для каждой из переменных x_i мы получим зависимость $x_i^{(k)}(t+H)$ от шага h_k , которую можно проинтерполировать в точку, соответствующую нулевому шагу $h_\infty = 0$. Ответ $x_i^{(\infty)}(t+H)$, полученный при нулевом шаге $h_\infty = 0$, мы рассматриваем как точный ответ для $x_i(t+H)$. Как и в алгоритме Ромберга, при прохождении отрезка $t \rightarrow t + H$ следует использовать схему второго порядка по шагу h_k и в качестве аргумента интерполируемой функции брать не сам шаг, а его квадрат h_k^2 .

3.2.1. Простейшие рецепты

В литературе существует большое разнообразие рекомендаций по выбору набора чисел N_k , по выбору схемы второго порядка, по алгоритму интерполяции в точку, соответствующую нулевому шагу.

Существует два наиболее популярных набора N_k :

$$N_k = \{4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128\}$$

и

$$N_k = \{4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24\}.$$

Хотя существуют задачи, для которых второй набор эффективнее первого, авторы горячо рекомендуют именно первый набор чисел. Это связано с тем, что он гораздо надежнее в тех случаях, когда происходят какие-либо «чудеса» с правой частью ОДУ.

Иногда предлагается использовать не обычную полиномиальную интерполяцию (см. часть I, раздел «Интерполяция»), а рациональную интерполяцию. Авторы категорически возражают против этой рекомендации. Разумеется, в большинстве задач оба метода дают практически совпадающие ответы, но есть задачи (например, с θ -функциями в правой части ОДУ), в которых использование рациональной интерполяции существенно уменьшает точность.

А теперь выясним, как должна выглядеть схема второго порядка для прохождения отрезка траектории $t \rightarrow t + H$ с шагом h_k .

«Классический» рецепт выглядит так:

$$\vec{x}(t + h_k) = \vec{x}(t) + h_k \vec{f}(\vec{x}(t)),$$

$$\vec{x}(t + 2h_k) = \vec{x}(t) + 2h_k \vec{f}(\vec{x}(t + h_k)),$$

$$\vec{x}(t + 3h_k) = \vec{x}(t + h_k) + 2h_k \vec{f}(\vec{x}(t + 2h_k)),$$

$$\vec{x}(t + 4h_k) = \vec{x}(t + 2h_k) + 2h_k \vec{f}(\vec{x}(t + 3h_k)),$$

...

$$\vec{x}(t + (N_k - 1)h_k) = \vec{x}(t + (N_k - 3)h_k) + 2h_k \vec{f}(\vec{x}(t + (N_k - 2)h_k)),$$

$$\vec{x}'(t + N_k h_k) = \vec{x}'(t + (N_k - 2)h_k) + 2h_k \vec{f}'(\vec{x}(t + (N_k - 1)h_k)),$$

$$\vec{x}(t + N_k h_k) \equiv \vec{x}(t + H) =$$

$$= \frac{1}{2} [\vec{x}'(t + N_k h_k) + \vec{x}(t + (N_k - 1)h_k) + h_k \vec{f}(\vec{x}'(t + N_k h_k))].$$

Этот несколько загадочный рецепт сначала нарушает заказанную точность (первый шаг делается не с точностью $O(h_k^2)$, а с точностью $O(h_k^1)$), а затем, в самом последнем шаге, восстанавливает заказанную точность. Дело в том, что

$$\begin{aligned} \vec{x}(t + (N_k - 1)h_k) &= 2\vec{x}(t + N_k h_k) - \vec{x}'(t + N_k h_k) - h_k \vec{f}(\vec{x}'(t + N_k h_k)) = \\ &= \vec{x}(t + N_k h_k) - h_k \vec{f}(\vec{x}'(t + N_k h_k)) + O(h_k^2), \end{aligned}$$

т.е. получившаяся схема инвариантна относительно замены $h_k \rightarrow -h_k$ и, следовательно, не содержит ошибки, пропорциональной нечетной степени $O(h_k^1)$. Специально подчеркнем, что последний шаг абсолютно необходим. Если его выкинуть, то точность катастрофически ухудшается.

Упрощенный рецепт гораздо проще:

$$\vec{x}'(t + h_k/2) = \vec{x}(t) + (h_k/2)\vec{f}(\vec{x}(t)),$$

$$\vec{x}(t + h_k) = \vec{x}(t) + h_k\vec{f}(\vec{x}'(t + h_k/2)),$$

$$\vec{x}(t + 2h_k) = \vec{x}(t) + 2h_k\vec{f}(\vec{x}(t + h_k)),$$

$$\vec{x}(t + 3h_k) = \vec{x}(t + h_k) + 2h_k\vec{f}(\vec{x}(t + 2h_k)),$$

...

$$\vec{x}(t + (N_k - 1)h_k) = \vec{x}(t + (N_k - 3)h_k) + 2h_k\vec{f}(\vec{x}(t + (N_k - 2)h_k)),$$

$$\vec{x}(t + N_k h_k) \equiv \vec{x}(t + H) = \vec{x}(t + (N_k - 2)h_k) + 2h_k\vec{f}(\vec{x}(t + (N_k - 1)h_k)),$$

он состоит из цепочки шагов схемы Рунге-Кутты второго порядка.

Поскольку упрощенный рецепт дает результаты ничуть не хуже, чем «классический», а в написании он удобнее, то мы рекомендуем именно его.

Последний вопрос, который необходимо обсудить, это вопрос о контроле точности.

Собственно, никаких особых усилий для контроля точности в данном алгоритме не требуется — правильный алгоритм полиномиальной интерполяции (см. часть I, раздел «Интерполяция») автоматически выдает оценку погрешности. Более того, в данном случае эта оценка есть попросту разница между результатами интерполяции на текущем k -ом этапе ($h_k = H/N_k$) и $(k - 1)$ -ом этапе ($h_{k-1} = H/N_{k-1}$).

Поэтому простейший способ реализации изложенных идей таков.

Алгоритм:

Выбираем постоянный шаг H по времени (как правило, следует брать H порядка 0.1 ... 0.3). Движемся по траектории с этим шагом. Каждый отдельный шаг реализуется так:

Для $k = 1, 2, \dots$ выполняем следующее:

Для очередного N_k из набора $\{4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128\}$ вычисляем шаг $h_k = H/N_k$.

С помощью данного шага h_k находим $\vec{x}_i^{(k)}(t + H)$:

$$\vec{x}'(t + h_k/2) = \vec{x}(t) + (h_k/2)\vec{f}(\vec{x}(t))$$

$$\vec{x}(t + h_k) = \vec{x}(t) + h_k\vec{f}(\vec{x}'(t + h_k/2))$$

$$\vec{x}(t + (n+1)h_k) = \vec{x}(t + (n-1)h_k) + 2h_k\vec{f}(\vec{x}(t + nh_k))$$

$$\vec{x}^{(k)}(t + H) \equiv \vec{x}(t + N_k h_k).$$

Для каждого x_i по k точкам:

$$\left(X_1 = h_1^2, Y_1 = x_i^{(1)}(t + H) \right),$$

$$\left(X_2 = h_2^2, Y_2 = x_i^{(2)}(t + H) \right),$$

...

$$\left(X_k = h_k^2, Y_k = x_i^{(k)}(t + H) \right)$$

выполняем полиномиальную интерполяцию зависимости $Y(X)$ в точку $X = 0$. Этот результат интерполяции, полученный на k -ом этапе, обозначаем $\tilde{x}_i^{(k)}(t + H)$.

Оцениваем погрешность $\vec{E}^{(k)}$ как

$$E_i^{(k)} = \tilde{x}_i^{(k)}(t + H) - \tilde{x}_i^{(k-1)}(t + H).$$

Превращаем вектор $\vec{E}^{(k)}$ в одно число $\epsilon^{(k)}$, характеризующее погрешность (см. раздел «Методы Рунге-Кутты»).

Если погрешность $\epsilon^{(k)}$ меньше заказанной, то полагаем $x_i(t + H) \equiv \tilde{x}_i^{(k)}(t + H)$ и завершаем данный шаг $t \rightarrow t + H$.

Предложенный тривиальный алгоритм особенно уместен в том случае, когда необходимо запомнить траекторию на отрезке времени $[t_1, t_2]$. Как правило, запоминать траекторию с очень мелким шагом нецелесообразно. Разумнее запомнить ее с шагом порядка $H = 0.1$, а затем при необходимости интерполировать в промежуточные точки.

3.2.2. Переменный шаг

Приведенный выше простейший рецепт требует подгонки параметра H . Дело в том, что для разных H сходимость будет достигаться на разных этапах — при малых H это может быть $k = 3$, а при больших H может понадобиться $k = 9$. Довольно ясно, что «коэффициент полезного действия» (производительность) пропорциональна $\eta_k = H/W_k$, где $W_k = \sum_{m=1}^k N_m$, поскольку величина W_k/H — это количество операций в расчете на единственный сдвиг по времени.

В среднем (для большинства задач, которые попадались авторам) оптимальное H соответствует $k = 6 \dots 8$. Однако для каждой конкретной задачи оптимальное H надо подбирать эмпирически. Это крайне нежелательно, особенно в том случае, когда поведение правой части ОДУ разное на разных отрезках траектории. Реализовать адаптивное изменение шага для интерполяционного алгоритма сложнее, чем для методов типа Рунге-Кутты,

поскольку здесь существуют две степени свободы: после очередного шага на H мы располагаем, во-первых, значением $\vec{x}(t + H)$, во-вторых, целым семейством погрешностей $\vec{E}^{(2)}, \vec{E}^{(3)}, \dots, \vec{E}^{(k)}$, где k — номер этапа, на котором была достигнута сходимость.

Разумеется, мы можем превратить векторы $\vec{E}^{(m)}$ в числа — в величины погрешностей $\epsilon^{(m)}$ (см. раздел «Методы Рунге-Кутты»). После этого для каждого m найти то значение шага $H_0^{(m)}$, которое при следующем шаге по времени обеспечит сходимость именно на m -ом этапе. Гипотетическая зависимость $\epsilon^{(m)}$ от h_m имеет вид $\epsilon^{(m)} = A_m h^{2m+1}$. Это связано с тем, что мы начинаем со схемы второго порядка, так что начальная погрешность пропорциональна $O(h^3)$. На каждом этапе интерполяции по h^2 из погрешности, которая представляет собой ряд по степеням h^2 , вычитается очередное лидирующее слагаемое, так что порядок схемы увеличивается на два. Поэтому, совершенно аналогично тому, как мы выбирали новый шаг для методов Рунге-Кутты:

$$H_0^{(m)} = 0.8H \left(\epsilon_0 / \epsilon^{(m)} \right)^{\frac{1}{(2m+1)}}.$$

Теперь возникает вопрос, какой именно шаг $H_0^{(m_0)}$ из всех этих возможных новых шагов $H_0^{(m)}$ следует выбрать.

Тривиальный ответ заключается в том, чтобы зафиксировать m_0 раз и навсегда, например, положить $m_0 = 6$. Так что упрощенный рецепт с адаптивным изменением шага состоит в следующем: мы всегда досчитываем до 6-го этапа (даже когда сходимость достигнута раньше), дальше 6-го этапа мы считаем только в том случае, когда $\epsilon^{(6)} > \epsilon_0$. После этого в качестве нового шага по времени мы берем $H = H_0^{(6)}$.

Самый правильный (на первый взгляд) ответ заключается в том, что мы выбираем m_0 , обеспечивающее максимальную производительность. Для этого вычисляем величины $\eta^{(m)} = H_0^{(m)} / W_m$ при $m = 1, 2, \dots, k$ и находим среди них максимальную $\eta^{(m_{\max})}$. После этого полагаем $m_0 = m_{\max}$.

К сожалению, реализовать эту идею на практике не так-то просто. Во-первых, найденное таким образом значение m_0 будет заведомо меньше k (мы ведь располагаем $H_0^{(m)}$ и $\eta^{(m)}$ только при $m = 1, 2, \dots, k$, так что $m_0 \leq k$). При этом истинное значение m_0 , обеспечивающее максимальную производительность, вполне может быть больше k , но мы не располагаем механизмом для определения этого m_0 и вычисления соответствующего $H_0^{(m_0)}$. Во-вторых, наши оценки для $H_0^{(m)}$ в общем-то не очень точны,

поэтому иногда оказывается, что максимальная производительность якобы достигается при несуразных значениях $m = 2$ или $m = 11$. Попытка использовать соответствующие $H_0^{(m)}$ немедленно приводит к значительному уменьшению производительности.

Итак, мы должны придумать способ, позволяющий выбрать $m_0 > k$. Для этого необходимо оценить соответствующее $H_0^{(m_0)}$. В принципе, есть рецепты для оценки гипотетических значений $H_0^{(k+1)}$ и $\eta^{(k+1)}$, однако эти рецепты не слишком надежны. Простейший выход заключается в том, что надо выполнять на один этап больше, чем нужно, т.е. после достижения заказанной точности совершать еще один оборот алгоритма.

Итак, «теоретически-оптимальный» рецепт выглядит следующим образом:

- мы никогда не выбираем m_0 равным крайним значениям (2 и 11), даже в том случае, когда $m_{\max}$ равно 2 или 11.
- мы всегда выполняем один лишний $(k + 1)$ -ый этап алгоритма, в результате чего находим величины $H_0^{(k+1)}$ и $\eta^{(k+1)}$. Кстати, это не так уж сильно снижает эффективность алгоритма, хотя мы делаем лишнюю работу, но и точность при этом заметно растет.
- мы выбираем m_0 равным $k - 1$, k или $k + 1$, т.е. m_0 должно отличаться от k не более чем на единицу. Если $\eta^{(k+1)} > \eta^{(k)}$, то мы полагаем $m_0 = k + 1$, если $\eta^{(k-1)} > \eta^{(k)}$, то мы полагаем $m_0 = k - 1$, в остальных случаях мы полагаем $m_0 = k$.

Как нам кажется, «возня» с переменным шагом в интерполяционных методах далеко не всегда оправдывается. Поэтому выбор между методом с постоянным шагом H , упрощенным методом с фиксированным m_0 и переменным шагом H , и «теоретически-идеальным» методом с переменными m_0 и H мы оставляем за читателем, в зависимости от его конкретной задачи.

Итак, интерполяционные методы безусловно следует применять для задач с относительно плавно меняющейся правой частью — они заметно точнее и гораздо производительнее методов Рунге-Кутты. Для задач с резко (и неаналитически) меняющейся правой частью их тоже можно применять, но при этом следует помнить, что для этих задач лучше подходят методы Рунге-Кутты.

3.3. Простейшие методы «предсказание-коррекция»

Как нам кажется, популярность этой группы методов совершенно необъяснима. Они заметно проигрывают как методам типа Рунге-Кутты, так

и интерполяционным методам, причем как по производительности, так и по точности. Единственным их достоинством является относительная простота написания. Тем не менее, мы (не слишком подробно) изложим эти методы. Делая мы это для того, чтобы не возникал соблазн их применить, если они встретятся Вам при чтении литературы.

Методы «предсказание-коррекция» делятся на две группы: методы с постоянным шагом (шаг фиксирован, и по ходу вычислений изменить его невозможно), эти методы мы для краткости будем называть схемами Адамса; и методы, в которых возможно адаптивное изменение шага, но которые в точности совпадают со схемами Адамса при постоянном шаге.

3.3.1. Схемы Адамса

Идея метода очень проста. Пусть мы движемся по траектории, причем шаг по времени h постоянный. Тогда, добравшись до n -й точки $\vec{x}(t_n)$, где $t_n = nh$, мы неизбежно должны были вычислить правые части ОДУ во всех предыдущих точках, в частности, в точках t_{n-1} и t_{n-2} . В дальнейшем мы везде будем писать $\vec{f}_n$ вместо $\vec{f}(\vec{x}(t_n))$. Следовательно, мы можем оценить старшие производные $\vec{x}(t)$ по времени в точке t_n , пользуясь значениями $\vec{f}_n$, $\vec{f}_{n-1}$ и $\vec{f}_{n-2}$:

$$\begin{aligned}\vec{f}_n &= \frac{d}{dt} \vec{x}(t_n), \\ \vec{f}_{n-1} &= \frac{d}{dt} \vec{x}(t_n) - h \frac{d^2}{dt^2} \vec{x}(t_n) + \frac{h^2}{2} \frac{d^3}{dt^3} \vec{x}(t_n) + \dots, \\ \vec{f}_{n-2} &= \frac{d}{dt} \vec{x}(t_n) - 2h \frac{d^2}{dt^2} \vec{x}(t_n) + \frac{(2h)^2}{2} \frac{d^3}{dt^3} \vec{x}(t_n) + \dots,\end{aligned}$$

откуда

$$\begin{aligned}\frac{d}{dt} \vec{x}(t_n) &= \vec{f}_n, \\ h \frac{d^2}{dt^2} \vec{x}(t_n) &\approx \frac{1}{2} \vec{f}_{n-2} - 2\vec{f}_{n-1} + \frac{3}{2} \vec{f}_n, \\ h^2 \frac{d^3}{dt^3} \vec{x}(t_n) &\approx \vec{f}_{n-2} - 2\vec{f}_{n-1} + \vec{f}_n.\end{aligned}$$

Следовательно, мы можем сделать «предсказание»:

$$\begin{aligned}\vec{x}'(t_{n+1}) &= \vec{x}(t_n) + h \frac{d}{dt} \vec{x}(t_n) + \frac{h^2}{2} \frac{d^2}{dt^2} \vec{x}(t_n) + \frac{h^3}{6} \frac{d^3}{dt^3} \vec{x}(t_n) + \dots \approx \\ &\approx \vec{x}(t_n) + \frac{h}{12} \left[5\vec{f}_{n-2} - 16\vec{f}_{n-1} + 23\vec{f}_n \right]\end{aligned}$$

с точностью до $O(h^3)$. Довольно ясно, что точность такого предсказания крайне невысока: вместо «прошупывания» траектории вперед и вычисления правой части ОДУ в промежуточных точках (как это делается в интерполяционных методах и методах Рунге-Кутты), мы пользуемся исключительно информацией с предыдущего отрезка траектории.

Положение до некоторой степени спасает второй шаг метода: мы вычисляем $\vec{f}'_{n+1}$ (правую часть ОДУ в точке $\vec{x}'(t_{n+1})$) и уточняем значения старших производных:

$$\begin{aligned} h \frac{d^2}{dt^2} \vec{x}(t_n) &= -\frac{1}{2} \vec{f}_{n-1} + \frac{1}{2} \vec{f}'_{n+1}, \\ h^2 \frac{d^3}{dt^3} \vec{x}(t_n) &= \vec{f}_{n-1} - 2\vec{f}_n + \vec{f}'_{n+1}, \end{aligned}$$

после чего проводим «коррекцию»:

$$\vec{x}(t_{n+1}) = \vec{x}(t_n) + \frac{h}{12} \left[-\vec{f}_{n-1} + 8\vec{f}_n + 5\vec{f}'_{n+1} \right].$$

Полученная точка считается очередной точкой на траектории.

Разумеется, реализовать этот алгоритм очень просто, достаточно просто запоминать значения правой части ОДУ в двух предыдущих точках. Впрочем, когда мы стартуем из начальной точки, двух предыдущих точек еще не существует, поэтому метод нуждается в двух «разгонных» шагах, которые обычно делаются методом Рунге-Кутты.

Как нам кажется, простота есть единственное достоинство изложенного метода. Во-первых, его точность — всего $O(h^3)$, причем увеличивать точность до $O(h^4)$ бессмысленно: непосредственная проверка показывает, что использование $\vec{f}_{n-3}$ отнюдь не улучшает ситуацию — довольно ясно, что t_{n-3} было слишком давно, чтобы иметь отношение к t_{n+1} . Во-вторых, на отрезке $[t_n, t_{n+1}]$ мы вычисляем правую часть ОДУ всего в двух точках, а даже в простейшем методе Рунге-Кутты четвертого порядка этих точек в два раза больше, так что надеяться на высокую точность не приходится. Поэтому неудивительно, что схемы Адамса при данной заказанной точности гораздо менее производительны, чем методы Рунге-Кутты или интерполяционные методы, причем это верно при любом (плавном или резко меняющемся) поведении правой части ОДУ. В третьих, хотя в методе и заложена естественная оценка погрешности (она равна разнице между $\vec{x}'(t_{n+1})$ и $\vec{x}(t_{n+1})$), адаптивное изменение шага реализовать невозможно — шаг по времени h фиксирован.

Обычно утверждается, что областью применения методов «предсказание-коррекция» являются задачи, в которых правая часть вычисляется

с исключительно большим трудом. В методах «предсказание-коррекция» минимальное количество вычислений правой части на единичный сдвиг по времени. Это действительно так, но и точность полученного ответа будет гораздо хуже, чем при применении других методов. Поэтому, как нам кажется, методы предсказание-коррекция следует применять в том случае, когда нас совершенно не интересует точность полученного ответа. Еще раз подчеркнем, что при данной заказанной точности методы Рунге-Кутты и интерполяционные методы требуют меньшего количества вычислений правой части на единичный сдвиг по времени, чем методы «предсказание-коррекция».

3.3.2. Переменный шаг

Методы «предсказание-коррекция» легко избавить от одного из их недостатков, а именно от фиксированного шага. Для этого достаточно запоминать не значения правой части ОДУ в двух предыдущих точках, а значения старших производных $\frac{d^2}{dt^2}\vec{x}(t_n)$ и $\frac{d^3}{dt^3}\vec{x}(t_n)$, вычисленные на предыдущих шагах. Разумеется, с помощью этих производных легко совершить новый шаг по времени h_n , который отнюдь не обязан совпадать с h_{n-1} .

Перепишем все соотношения для схем Адамса в терминах четырех величин $\frac{d^m}{dt^m}\vec{x}(t_n)$, где $m = 0, 1, 2, 3$. К n -ому шагу мы располагаем величинами:

$$\begin{aligned}\vec{x}(t_n), \\ \frac{d}{dt}\vec{x}(t_n) = \vec{f}_n, \\ h\frac{d^2}{dt^2}\vec{x}(t_n) = \frac{1}{2}\vec{f}_{n-2} - 2\vec{f}_{n-1} + \frac{3}{2}\vec{f}_n, \\ h^2\frac{d^3}{dt^3}\vec{x}(t_n) = \vec{f}_{n-2} - 2\vec{f}_{n-1} + \vec{f}_n.\end{aligned}$$

Выполняем «предсказание»:

$$\begin{aligned}\vec{x}'(t_{n+1}) &= \vec{x}(t_n) + h\frac{d}{dt}\vec{x}(t_n) + \frac{h^2}{2}\frac{d^2}{dt^2}\vec{x}(t_n) + \frac{h^3}{6}\frac{d^3}{dt^3}\vec{x}(t_n), \\ \frac{d}{dt}\vec{x}'(t_{n+1}) &= \frac{d}{dt}\vec{x}(t_n) + h\frac{d^2}{dt^2}\vec{x}(t_n) + \frac{h^2}{2}\frac{d^3}{dt^3}\vec{x}(t_n), \quad (*) \\ \frac{d^2}{dt^2}\vec{x}'(t_{n+1}) &= \frac{d^2}{dt^2}\vec{x}(t_n) + h\frac{d^3}{dt^3}\vec{x}(t_n), \\ \frac{d^3}{dt^3}\vec{x}'(t_{n+1}) &= \frac{d^3}{dt^3}\vec{x}(t_n).\end{aligned}$$

После этого вычисляем $\vec{f}'_{n+1} \equiv \vec{f}'(\vec{x}'(t_{n+1}))$. Это позволяет нам произвести второй шаг («коррекцию»), при котором величину

$$\vec{x}'(t_{n+1}) = \vec{x}(t_n) + \frac{h}{12} [5\vec{f}_{n-2} - 16\vec{f}_{n-1} + 23\vec{f}_n]$$

надо заменить на

$$\vec{x}(t_{n+1}) = \vec{x}(t_n) + \frac{h}{12} [-\vec{f}_{n-1} + 8\vec{f}_{n-1} + 5\vec{f}'_{n+1}],$$

т. е. добавить к ней

$$\frac{5h}{12} [\vec{f}'_{n+1} - 3\vec{f}_n + 3\vec{f}_{n-1} - \vec{f}_{n-2}] = \frac{5h}{12} \Delta^3 \vec{f}_n.$$

Как и следовало ожидать, поправка пропорциональна третьей разностной производной $\vec{f}'$ (мы обозначили ее $\Delta^3 \vec{f}_n$). Действительно, раз мы работаем с точностью до $O(h^3)$, то разница между двумя ответами для одной и той же величины должна быть пропорциональна четвертой разностной производной $\vec{x}$, которая совпадает с третьей разностной производной $\vec{f}$. Ясно, что поправки в остальных $\frac{d^m}{dt^m} \vec{x}(t_{n+1})$ также будут пропорциональны $\Delta^3 \vec{f}_n$. Действительно, поправка в $h^2 \frac{d^3}{dt^3} \vec{x}(t_n)$:

$$\begin{aligned} h^2 \frac{d^3}{dt^3} \vec{x}(t_{n+1}) - h^2 \frac{d^3}{dt^3} \vec{x}'(t_{n+1}) &= [\vec{f}'_{n+1} - 2\vec{f}_n + \vec{f}_{n-1}] - \\ &- [\vec{f}_n - 2\vec{f}_{n-1} + \vec{f}_{n-2}] = \Delta^3 \vec{f}_n. \end{aligned}$$

Совершенно аналогично, поправка в $h \frac{d^2}{dt^2} \vec{x}(t_n)$ равна:

$$\begin{aligned} h \frac{d^2}{dt^2} \vec{x}(t_{n+1}) - h \frac{d^2}{dt^2} \vec{x}'(t_{n+1}) &= \left[\frac{3}{2} \vec{f}'_{n+1} - 2\vec{f}_n + \frac{1}{2} \vec{f}_{n-1} \right] - \\ &- \left[\frac{3}{2} \vec{f}_n - 2\vec{f}_{n-1} + \frac{1}{2} \vec{f}_{n-2} + (\vec{f}_n - 2\vec{f}_{n-1} + \vec{f}_{n-2}) \right] = \frac{3}{2} \Delta^3 \vec{f}_n. \end{aligned}$$

И, наконец, поправка в $\frac{d}{dt} \vec{x}(t_n)$:

$$\begin{aligned} \frac{d}{dt} \vec{x}(t_{n+1}) - \frac{d}{dt} \vec{x}'(t_{n+1}) &= [\vec{f}'_{n+1}] - \\ &- \left[\vec{f}_n + \left(\frac{3}{2} \vec{f}_n - 2\vec{f}_{n-1} + \frac{1}{2} \vec{f}_{n-2} \right) + (\vec{f}_n - 2\vec{f}_{n-1} + \vec{f}_{n-2}) \right] = \Delta^3 \vec{f}_n. \end{aligned}$$

Заметим, что, как только мы вычислим $\vec{f}'_{n+1}$, то немедленно получим значение этой последней поправки. Она равна разности $f'_{n+1} - \frac{d}{dt}x'(t_{n+1})$. Следовательно, вычислив $\vec{f}'_{n+1}$, мы тут же находим величину $\Delta^3 \vec{f}_n$. После этого можно внести поправки во все остальные $\frac{d^m}{dt^m} \vec{x}(t_{n+1})$.

Так что окончательный рецепт очень прост:

- Располагая четырьмя величинами $\frac{d^m}{dt^m} \vec{x}(t_n)$, мы, пользуясь (*), находим $\frac{d^m}{dt^m} \vec{x}'(t_{n+1})$ («предсказание»).
- Затем вычисляем $\vec{f}'_{n+1} = \vec{f}'(\vec{x}'(t_{n+1}))$. Находим $\Delta^3 \vec{f}_n = \vec{f}'_{n+1} - \frac{d}{dt} \vec{x}'(t_{n+1})$.
- После чего вносим поправки в $\frac{d^m}{dt^m} \vec{x}'(t_{n+1})$ («коррекция»):

$$\vec{x}(t_{n+1}) = \vec{x}'(t_{n+1}) + \frac{5h}{12} \Delta^3 \vec{f}_n,$$

$$\frac{d}{dt} x(t_{n+1}) = \vec{f}'_{n+1},$$

$$h \frac{d^2}{dt^2} x(t_{n+1}) = h \frac{d^2}{dt^2} \vec{x}'(t_{n+1}) + \frac{3}{2} \Delta^3 \vec{f}_n,$$

$$h^2 \frac{d^3}{dt^3} x(t_{n+1}) = h^2 \frac{d^3}{dt^3} \vec{x}'(t_{n+1}) + \Delta^3 \vec{f}_n.$$

Выбор h для следующего шага по времени реализуется так же, как в методе Рунге-Кутты. Действительно, в качестве погрешности $\vec{E}$ можно рассматривать $\vec{E} = \frac{5h}{12} \Delta^3 \vec{f}_n$, причем она пропорциональна четвертой степени h : $\vec{E} = \vec{A}h^4$.

Как и схемы Адамса, этот метод нуждается в двух «разгонных» шагах, которые можно сделать методом Рунге-Кутты.

Разумеется, при адаптивном изменении шага результаты получаются лучше, чем при постоянном шаге, но все равно этот метод существенно проигрывает как методам Рунге-Кутты, так и интерполяционным методам.

Схемы Адамса и их аналоги с переменным шагом лучше не использовать.

4. Краевая задача для обыкновенных дифференциальных уравнений

Вариантов краевой задачи для системы ОДУ существует великое множество. Тем не менее все они (с некоторой натяжкой) сводятся к такой постановке задачи: существует система N обыкновенных дифференциальных

уравнений для N неизвестных, которая рассматривается на конечном или бесконечном интервале $[t_1, t_2]$. Мы располагаем M граничными условиями на левом конце интервала:

$$\mathcal{L}_k(\vec{x}(t_1)) = 0, \quad k = 1, \dots, M$$

и, соответственно, $N - M$ граничными условиями на правом конце интервала:

$$\mathcal{R}_k(\vec{x}(t_2)) = 0, \quad k = 1, \dots, N - M.$$

Необходимо найти решение системы ОДУ, удовлетворяющее этим условиям. Задачи на спектр дифференциальных операторов также можно описать в этих терминах, если ввести лишнюю переменную x_{N+1} , равную искомому собственному значению. Переменная x_{N+1} должна удовлетворять тривиальному уравнению движения $\frac{d}{dt}x_{N+1} = 0$.

4.1. Метод «стрельбы»

Метод стрельбы не только самый тривиальный, но и самый эффективный метод решения краевой задачи. К сожалению, он далеко не всегда работает. Общая же рекомендация очень проста: если стрелять можно, то нужно стрелять.

4.1.1. Общий рецепт

Пусть $M \geq N/2$. Тогда прежде всего следует параметризовать решение M уравнений $\mathcal{L}_k(\vec{x}(t_1)) = 0$ с помощью $N - M$ параметров α_k , т. е. найти такие $x_i(\alpha_1, \dots, \alpha_{N-M})$, что $\mathcal{L}_k(\vec{x}(\alpha_1, \dots, \alpha_{N-M})) = 0$ для всех k .

Часто встречается рекомендация просто выразить какие-то M штук $x_i(t_1)$ через остальные $N - M$ штук $x_i(t_1)$ с помощью уравнений $\mathcal{L}_k$. Этой рекомендации лучше не следовать. Например, если граничное условие имеет вид $x_1^2 + x_2^2 = 1$, то полагать $x_2 = \pm\sqrt{1 - x_1^2}$ весьма неразумно. Во-первых, мы получаем неоднозначное выражение. Во-вторых, производная dx_2/dx_1 бесконечна при $x_1 = \pm 1$. Куда разумнее ввести параметр α и записать решение уравнения $x_1^2 + x_2^2 = 1$ в виде: $x_1 = \cos(\alpha)$, $x_2 = \sin(\alpha)$.

Каждый фиксированный набор $(\alpha_1, \dots, \alpha_{N-M})$ даст нам вектор $\vec{x}(\alpha_1, \dots, \alpha_{N-M})$, который удовлетворяет граничным условиям $\mathcal{L}_k$ на левом конце интервала. Будем рассматривать этот вектор как начальное условие для задачи Коши на интервале $[t_1, t_2]$: $\vec{x}(t_1, \alpha_1, \dots, \alpha_{N-M}) \equiv \vec{x}(\alpha_1, \dots, \alpha_{N-M})$. Решим эту задачу Коши, т. е. «произведем выстрел». В результате мы получим значения $\vec{x}(t_2, \alpha_1, \dots, \alpha_{N-M})$ на правом конце интервала.

Разумеется, результат этого «выстрела», скорее всего, будет неудачным: величины $\vec{x}(t_2, \alpha_1, \dots, \alpha_{N-M})$, почти наверняка, не будут удовлетворять граничным условиям $\mathcal{R}_k$ на правом конце интервала. Если бы не граничные условия $\mathcal{R}_k$, полученное решение задачи Коши было бы и решением краевой задачи, ведь $\vec{x}(t, \alpha_1, \dots, \alpha_{N-M})$ удовлетворяет граничным условиям на левом конце интервала и является решением системы ОДУ. Однако условия $\mathcal{R}_k$ все портят: почти наверняка, $\mathcal{R}_k(\vec{x}(t_2, \alpha_1, \dots, \alpha_{N-M})) \neq 0$, иными словами, «выстрел не попадает в цель».

Следовательно, чтобы решить нашу краевую задачу, нам надо так подобрать параметры $\alpha_1, \dots, \alpha_{N-M}$ («прицелиться»), чтобы $\vec{x}(t, \alpha_1, \dots, \alpha_{N-M})$ удовлетворяло граничным условиям при $t = t_2$. Иными словами, нам надо решить систему $N - M$ уравнений

$$\mathcal{R}_1(\vec{x}(t_2, \alpha_1, \dots, \alpha_{N-M})) = 0, \dots, \mathcal{R}_{N-M}(\vec{x}(t_2, \alpha_1, \dots, \alpha_{N-M})) = 0$$

для $N - M$ неизвестных $\alpha_1, \dots, \alpha_{N-M}$. При этом мы не располагаем явным аналитическим выражением для $\mathcal{R}_k(\vec{x}(t_2, \alpha_1, \dots, \alpha_{N-M}))$. Действительно, чтобы вычислить $\mathcal{R}_k$ для каждого данного набора $\alpha_1, \dots, \alpha_{N-M}$, необходимо решить задачу Коши на интервале $[t_1, t_2]$.

Решать систему уравнений $\mathcal{R}_k$ относительно неизвестных α_i придется методом Ньютона (см. часть I, раздел «Многомерные корни»). Именно, выбрав некоторую начальную точку $\alpha_1^{(0)}, \dots, \alpha_{N-M}^{(0)}$, мы вычисляем, во-первых, соответствующие значения $\mathcal{R}_k$, и, во-вторых, значения производных $\partial_{\alpha_i} \mathcal{R}_k$. К сожалению, производные в данном случае можно вычислить только совершенно варварским методом варьирования по α_i :

$$\partial_{\alpha_i} \mathcal{R}_k = \frac{1}{\epsilon} \left[\mathcal{R}_k(\vec{x}(t_2, \alpha_1, \dots, \alpha_i + \epsilon, \dots, \alpha_{N-M})) - \mathcal{R}_k(\vec{x}(t_2, \alpha_1, \dots, \alpha_i, \dots, \alpha_{N-M})) \right],$$

причем ϵ нельзя брать ни слишком большим (это будет уже не производная), ни слишком маленьким (в этом случае мы найдем не производную, а разность в ошибках округления). Оптимальное значение ϵ зависит от конкретной задачи. Вычислив производные, мы решаем систему линейных уравнений для смещений $\delta\alpha_i$, которые позволят получить следующее приближение к корню $\alpha_i^{(1)} = \alpha_i^{(0)} + \delta\alpha_i$:

$$\mathcal{R}_k + \partial_{\alpha_i} \mathcal{R}_k \delta\alpha_i = 0.$$

Напомним, что следует установить некоторую максимальную длину $\Delta_{\max}$ для вектора $\delta\vec{\alpha}$. Если $\delta\vec{\alpha}$ длиннее $\Delta_{\max}$, то направление вектора сохраняется, а длина укорачивается до $\Delta_{\max}$. Опять-таки, выбор $\Delta_{\max}$ зависит от

конкретной задачи. Заметим, что только для того, чтобы найти следующее приближение к корню $\alpha_i^{(1)}$, нам приходится $(N - M + 1)$ раз решать задачу Коши на интервале $[t_1, t_2]$. Процедура повторяется до тех пор, пока не будет с достаточной точностью найден искомым корень системы уравнений $\mathcal{R}_k(\vec{x}(t_2, \alpha_1, \dots, \alpha_{N-M})) = 0$, что и завершит решение краевой задачи.

В том случае, когда $M \leq N/2$, мы просто меняем местами левый и правый концы интервала, т.е. «стреляем» не слева направо, а справа налево. При этом начальной точкой для задачи Коши является t_2 , а конечной — точка t_1 .

К сожалению, далеко не всегда системе ОДУ все равно, в какую сторону по времени t мы движемся (см., например, следующий раздел). Поэтому часто возникают ситуации, когда приходится «стрелять» из обеих концевых точек в направлении некоторой промежуточной точки $t_3 \in [t_1, t_2]$. При этом решение левых граничных условий мы параметризуем $N - M$ параметрами $\alpha_1, \dots, \alpha_{N-M}$, решаем задачу Коши на интервале $[t_1, t_3]$ («стрельба» слева направо) и получаем точку $\vec{x}_L(t_3, \alpha_1, \dots, \alpha_{N-M})$. Аналогично, решение правых граничных условий параметризуем M параметрами $\alpha_{N-M+1}, \dots, \alpha_N$, решаем задачу Коши на интервале $[t_2, t_3]$ («стрельба» справа налево) и получаем точку $\vec{x}_R(t_3, \alpha_{N-M+1}, \dots, \alpha_N)$. После чего необходимо решить систему N уравнений $\vec{x}_L(t_3, \alpha_1, \dots, \alpha_{N-M}) = \vec{x}_R(t_3, \alpha_{N-M+1}, \dots, \alpha_N)$ с N неизвестными $\alpha_1, \dots, \alpha_N$.

Еще раз напоминаем, что «стрельба» есть самый эффективный способ решения краевой задачи, если только ее удастся применить к данной системе ОДУ.

4.1.2. Уравнение типа Шредингера

Формально говоря, задача поиска дискретного спектра квантовомеханической системы решается общими методами, описанными в предыдущем разделе. Однако, при решении этой задачи необходимо учитывать столько специфических деталей, что она требует отдельного обсуждения.

Итак, пусть мы ищем уровни энергии с помощью уравнения типа Шредингера⁶:

$$\psi''(x) + (E\rho(x) - V(x))\psi(x) = 0.$$

Ранее мы обозначали переменную, по которой идет дифференцирование, буквой $\langle t \rangle$ и называли ее «временем». Для уравнений Шредингера это было

⁶Напомним, что большинство задач о спектре дифференциального оператора второго порядка можно свести к этому виду. Достаточно провести замену переменной x и переопределить функцию ψ . Более того, можно даже добиться, чтобы из уравнения исчез вес: $\rho(x)$ можно сделать равным 1.

бы нелепо, так что переменная теперь обозначается буквой « x » и называется координатой.

Обычно задача ставится на бесконечном интервале $-\infty < x < \infty$. Именно этот случай мы в основном и будем рассматривать. Задача также может быть поставлена на конечном интервале $[x_1, x_2]$ с определенными граничными условиями, например $\psi(x_1) = 0 = \psi(x_2)$, что соответствует бесконечно высоким стенкам на концах интервала. Этот случай нет смысла рассматривать отдельно, поскольку для него вполне применимы (с незначительной модификацией) все рекомендации, относящиеся к бесконечному интервалу. Наконец, задача может быть поставлена на конечном интервале $[x_1, x_2]$, но с сингулярным поведением функций $V(x)$ или $\rho(x)$ на одном из концов интервала (или на обоих концах интервала). Этот случай мы рассмотрим отдельно.

В большинстве реальных задач потенциал или растет на бесконечности (как у осциллятора), или выходит на константу (потенциальная яма). Следовательно (при не слишком патологическом поведении функции $\rho(x)$ при $x \rightarrow \pm\infty$), условие квадратичной интегрируемости с весом $\rho(x)$ означает убывание волновой функции $\psi(x)$ при $x \rightarrow \pm\infty$, причем убывание должно быть не хуже экспоненциального. Уравнение линейное, так что у него существует два линейно независимых решения, $\psi_1(x)$ и $\psi_2(x)$, вронскиан которых есть константа:

$$w(\psi_1, \psi_2) = \psi_1'(x)\psi_2(x) - \psi_1(x)\psi_2'(x) = \text{const.}$$

Если одно из решений экспоненциально (или быстрее) убывает на бесконечности, то второе, соответственно, растет.

Довольно ясно, что если мы численно решаем задачу Копи для нашего уравнения в той области, которую мы считаем эффективной бесконечностью, причем в направлении слева направо, то довольно скоро останется только растущее слева направо решение. Это произойдет вне зависимости от начальных условий: даже если начальные условия соответствовали убывающему решению, ошибки округления очень скоро породят растущее решение. Совершенно то же самое произойдет, если решать задачу в этой же области, но в направлении справа налево, только решение будет расти «наоборот» — оно будет расти справа налево. Поэтому теми рецептами, в которых рекомендуют «стрелять» из эффективной отрицательной бесконечности в эффективную положительную бесконечность (или наоборот), лучше не пользоваться. Безусловно, есть потенциалы, в которых эти рецепты дают неплохую точность, но лучше не искушать судьбу.

Из сказанного очевидно, что надежнее всего «стрелять» из обеих бесконечностей навстречу друг другу и сшивать получившиеся решения где-то

посередине, в точке X_0 . Более того, легко проверить, что в этом случае можно даже не заботиться о правильном соотношении между функцией и ее производной на эффективной бесконечности. Это верно практически всегда, разве что убывание волновой функции на бесконечности уж очень медленное (когда энергия уровня приблизительно равна значению потенциала на бесконечности). Так что можно было бы брать совершенно несурзные начальные условия на концах интервала $[X_-, X_+]$, где X_+ и X_- изображают $+\infty$ и $-\infty$ соответственно. Например, можно полагать $\psi(X_-) = 1$, $\psi'(X_-) = 0$ или $\psi(X_-) = 0$, $\psi'(X_-) = 1$. Аналогично, можно полагать $\psi(X_+) = 1$, $\psi'(X_+) = 0$ или $\psi(X_+) = 0$, $\psi'(X_+) = 1$. Несмотря на явно неправильные граничные условия, мы, скорее всего, получим совершенно правильный ответ для уровней энергии.

Надежнее, однако, брать правильное (квазиклассическое) соотношение между ψ и ψ' : $\psi'(X_-)/\psi(X_-) = \sqrt{V(X_-) - E}$ (убывание при $x \rightarrow -\infty$) и $\psi'(X_+)/\psi(X_+) = -\sqrt{V(X_+) - E}$ (убывание при $x \rightarrow +\infty$). Более того, чтобы застраховаться от возможного переполнения при стрельбе (это особенно существенно для низколежащих уровней, для которых $\sqrt{V(X_{\pm}) - E}$ велико, и волновая функция ψ очень быстро растет в направлении от краев интервала $[X_-, X_+]$ к его середине), следует брать величины $\psi(X_{\pm})$ достаточно маленькими, скажем, $\psi(X_{\pm}) = 10^{-100}$.

Благодаря линейности уравнения Шредингера, мы получим не два условия в средней точке X_0 (формально говоря, следовало бы требовать совпадения ψ и совпадения ψ' в точке X_0), а только одно. Действительно, решая задачу Коши на интервале $[X_-, X_0]$ с начальными условиями в точке X_- : $\psi(X_-) = 10^{-100}$, $\psi'(X_-) = \psi(X_-)\sqrt{V(X_-) - E}$, мы получаем решение $\psi_-(x)$, убывающее при $x \rightarrow -\infty$. Аналогично, решая задачу Коши на интервале $[X_+, X_0]$ с начальными условиями в точке X_+ : $\psi(X_+) = 10^{-100}$, $\psi'(X_+) = -\psi(X_+)\sqrt{V(X_+) - E}$, мы получаем решение $\psi_+(x)$, убывающее при $x \rightarrow +\infty$. Если два эти решения сшиваются в точке X_0 , т. е. если $\psi'_-(X_0)/\psi_-(X_0) = \psi'_+(X_0)/\psi_+(X_0)$, то E есть уровень энергии. Разумеется, это условие немедленно переписывается как

$$w(\psi_-, \psi_+) = \psi'_-(X_0)\psi_+(X_0) - \psi'_+(X_0)\psi_-(X_0) = 0,$$

т. е. как условие обращения вронскиана функций ψ_- и ψ_+ в ноль. Кстати, при вычислении вронскиана надо проверять, чтобы не произошло переполнения или недополнения в каждом из произведений.

Итак, задача о дискретном спектре сводится попросту к поиску корней функции $w(\psi_-, \psi_+)$, зависящей от одного аргумента E (см. часть I, раздел «Одномерные корни»). Для потенциала типа потенциальной ямы корни

необходимо найти на интервале $\min_x V(x) \leq E \leq \min\{V(-\infty), V(+\infty)\}$. Для потенциала типа осциллятора энергия ограничена только снизу: $E \geq \min_x V(x)$.

Естественный способ контроля точности заключается в том, что следует сделать эффективную бесконечность «еще более бесконечной», т.е. увеличить X_- и X_+ раза в полтора. Одновременно следует увеличить (скажем, на порядок) точность, с которой мы решаем задачу Коши. Если уровни почти не сдвинутся, то есть надежда, что они найдены правильно. Дополнительная проверка заключается в том, что надо следить за количеством корней волновой функции: в соответствии с осцилляционной теоремой номер уровня совпадает с числом корней (если нумерация уровней начинается с нуля).

Первый шаг при поиске корней вронскиана в любом случае заключается в первоначальной табуляции вронскиана в том диапазоне энергий, в котором мы ищем уровни. Следует понимать, что когда мы ищем уровень энергии E , который близок к верхнему порогу ямы (когда E близко к $V(+\infty)$ или $V(-\infty)$), то необходимо очень далеко отодвинуть эффективную бесконечность. В то же время для низлежащих уровней (когда E близко к $\min_x V(x)$) вполне можно использовать не очень большие X_- и X_+ . Поэтому при табуляции по мере роста E стоит постепенно отодвигать X_- и X_+ . Эта рекомендация относится и к потенциалам осцилляторного типа: ведь для большинства потенциалов чем больше E , тем больше размер области, в которой локализована волновая функция.

Теперь обсудим поиск уровней для системы уравнений типа Шредингера $N \times N$:

$$\psi_i''(x) + (E\rho(x) - V_{ij}(x))\psi_j(x) = 0,$$

по индексу j здесь идет суммирование от 1 до N . В принципе все рекомендации, приведенные для одного уравнения, сохраняют силу. Для того случая, когда обе матрицы

$$E\rho(-\infty) - V_{ij}(-\infty)$$

и

$$E\rho(+\infty) - V_{ij}(+\infty)$$

пропорциональны единичной матрице (как ни странно, на практике чаще всего встречается именно этот частный случай), вообще никаких проблем не возникает. Действительно, в этом случае все N линейно независимых

решений, которые убывают при $x \rightarrow -\infty$, убывают с одинаковой скоростью. Благодаря этому, их довольно легко построить. Первое из решений $\psi_i^{(1)-}(x)$ мы строим, полагая, что на отрицательной бесконечности отлична от нуля только первая компонента волновой функции:

$$\psi_1^{(1)-}(X_-) = 10^{-100}, \quad \frac{d}{dx} \psi_1^{(1)-}(X_-) = \kappa \psi_1^{(1)-}(X_-),$$

$$\psi_{i \neq 1}^{(1)-}(X_-) = 0, \quad \frac{d}{dx} \psi_{i \neq 1}^{(1)-}(X_-) = 0,$$

здесь

$$\kappa \equiv \sqrt{V_{11}(X_-) - E} = \sqrt{V_{22}(X_-) - E} = \dots = \sqrt{V_{NN}(X_-) - E}.$$

Совершенно аналогично строится второе решение $\psi_-^{(2)}(x)$ и так далее. Поскольку при «стрельбе» из X_- в X_0 все эти решения растут с одинаковой скоростью, то нет риска, что произойдет вырождение, т. е. нет риска, что разные начальные условия приведут к одной и той же (самой быстрорастущей) функции.

То же самое делаем при «стрельбе» из положительной бесконечности, т. е. из X_+ в X_0 . Получаем два семейства решений $\psi_i^{(k)-}(x)$ и $\psi_i^{(k)+}(x)$. Если при данном E некоторая линейная комбинация «левых» решений $\psi_i^{(k)-}(x)$ сплывается с некоторой линейной комбинацией «правых» решений $\psi_i^{(k)+}(x)$ в точке X_0 , то мы нашли уровень энергии. Иными словами, нам надо найти такие E , чтобы детерминант $w(E)$:

$$\begin{vmatrix} \psi_1^{(1)-}(X_0) & \dots & \psi_1^{(N)-}(X_0) & \psi_1^{(1)+}(X_0) & \dots & \psi_1^{(N)+}(X_0) \\ \psi_2^{(1)-}(X_0) & \dots & \psi_2^{(N)-}(X_0) & \psi_2^{(1)+}(X_0) & \dots & \psi_2^{(N)+}(X_0) \\ \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ \psi_N^{(1)-}(X_0) & \dots & \psi_N^{(N)-}(X_0) & \psi_N^{(1)+}(X_0) & \dots & \psi_N^{(N)+}(X_0) \\ \psi_1^{(1)-'}(X_0) & \dots & \psi_1^{(N)-'}(X_0) & \psi_1^{(1)+'}(X_0) & \dots & \psi_1^{(N)+'}(X_0) \\ \psi_2^{(1)-'}(X_0) & \dots & \psi_2^{(N)-'}(X_0) & \psi_2^{(1)+'}(X_0) & \dots & \psi_2^{(N)+'}(X_0) \\ \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ \psi_N^{(1)-'}(X_0) & \dots & \psi_N^{(N)-'}(X_0) & \psi_N^{(1)+'}(X_0) & \dots & \psi_N^{(N)+'}(X_0) \end{vmatrix}$$

обращался в ноль. Иными словами, речь опять-таки идет всего лишь о поиске корней функции $w(E)$, зависящей от одного аргумента E .

Случай, когда матрицы $E\rho(\pm\infty) - V_{ij}(\pm\infty)$ имеют невырожденный спектр, представляет гораздо большие технические трудности. Формально говоря, при невырожденном спектре мы также с легкостью можем построить оба семейства решений $\psi_i^{(k)-}(x)$ и $\psi_i^{(k)+}(x)$. Действительно, находим $\lambda^{(1)}$ — первое из собственных значений матрицы $E\rho(-\infty) - V_{ij}(-\infty)$ и соответствующий собственный вектор $\vec{c}^{(1)}$. После этого полагаем

$$\begin{aligned}\psi_i^{(1)-}(X_-) &= 10^{-100} \times c_i^{(1)}, \\ \frac{d}{dx}\psi_i^{(1)-}(X_-) &= \kappa\psi_i^{(1)-}(X_-)\end{aligned}$$

где $\kappa = \sqrt{\lambda^{(1)}}$. Совершенно аналогично строим $\psi_i^{(2)-}(x)$ и так далее.

Но при этом следует понимать, что если мы слишком далеко отодвинем X_- , то, начав с заведомо линейно независимых решений в точке X_- , мы, дойдя до точки X_0 , можем получить одну и ту же функцию $\psi_i^{(m)-}(x)$. Это будет функция, соответствующая максимальному $\lambda^{(m)}$. Функция $\psi_i^{(m)-}(x)$ родится просто за счет ошибок округления и немедленно начнет расти гораздо быстрее, чем все остальные функции. Если X_- достаточно велико, то при $x = X_0$ только одна $\psi_i^{(m)-}(x)$ и останется — все остальные функции по сравнению с нею будут машинным нулем. Это обстоятельство фатально скажется на точности нахождения уровней. Поэтому при невырожденном спектре матриц $E\rho(\pm\infty) - V_{ij}(\pm\infty)$ возникает дополнительное существенное ограничение сверху на значения эффективной бесконечности $X_{\pm}$. Для обычного уравнения Шредингера слишком большие $X_{\pm}$ означают лишь много лишней работы и высокую вероятность переполнения, но не приводят к потере точности. Здесь же слишком большие $X_{\pm}$ могут приводить к неточно найденным, а то и вовсе неправильным уровням.

4.1.3. Особые точки

Как показывает печальный опыт, практически любой человек, сталкиваясь с особой точкой в уравнении Шредингера, начинает с того, что пытается решить задачу «в лоб». Потом обнаруживает, что найденный таким способом спектр атома водорода не очень похож на правильный. И лишь тогда он понимает, что сингулярная функция на равномерной решетке со сколь угодно малым шагом непригодна для численного счета. Действительно, рассмотрим функцию $f(r) = 1/r$ в окрестности точки $r = 0$. На равномерной решетке со сколь угодно малым шагом h она ведет себя как

$f(h) = h^{-1}$, $f(2h) = h^{-1}/2$, $f(3h) = h^{-1}/3$, $f(4h) = h^{-1}/4$, ..., что не слишком похоже на гладкую зависимость.

Между тем правильный рецепт очень прост: задачу следует превратить в регулярную с помощью тривиальной замены переменной. Например, если речь идет о радиальном уравнении Шредингера для атома водорода

$$\frac{d^2}{dr^2} R(r) + \left(E + \frac{\alpha}{r} - \frac{l(l+1)}{r^2} \right) R(r) = 0,$$

то следует перейти от радиальной переменной r к новой переменной

$$\rho \equiv r + A \log r,$$

которая меняется в диапазоне от $-\infty$ до $+\infty$. Величина A здесь — некоторая константа. Введем равномерную сетку по ρ . Тогда при $r \rightarrow \infty$ мы получим почти равномерную сетку по r . В то же время при $r \rightarrow 0$ произойдет существенное сгущение сетки. Поэтому можно надеяться, что в терминах ρ задача станет регулярной. Непосредственная проверка подтверждает эту гипотезу:

$$\frac{d^2}{d\rho^2} R(\rho) - \frac{A}{(r+A)^2} \frac{d}{d\rho} R(\rho) + \left(\frac{Er^2}{(r+A)^2} + \frac{\alpha r}{(r+A)^2} - \frac{l(l+1)}{(r+A)^2} \right) R(\rho) = 0.$$

В принципе можно еще и восстановить эрмитовость задачи заменой $R(\rho) = \tilde{R}(\rho) \sqrt{r/(r+A)}$ (эта замена истребит слагаемое с первой производной по ρ), но делать это необязательно. Значение константы A подбирается в зависимости от того, в каком диапазоне мы ищем уровни и каковы конкретные значения параметров l и α . Обычно $A = 0.1 \dots 1.0$.

Итак, мы получили регулярную задачу, которая подробно обсуждена в предыдущем разделе. Единственным неудобством является отсутствие явного аналитического выражения для обратной замены $\rho \rightarrow r$ (а в окончательное уравнение входит как раз $r(\rho)$). Можно, конечно, искать r_0 для каждого данного ρ_0 , решая уравнение $\rho_0 = r_0 + A \log r_0$, но разумнее раз и навсегда протабулировать зависимость $r(\rho)$ на достаточно частой сетке и при необходимости интерполировать ее в промежуточные точки.

Если в уравнение входит сингулярность более высокого порядка, например β/r^4 , то ее следует истребить заменой

$$\rho \equiv r - A/r.$$

Эта замена хороша еще и тем, что существует явное аналитическое выражение для $r(\rho)$. В общем-то, несмотря на «превышение сингулярности», эту замену можно применять и для задач с сингулярностью $1/r^2$.

Очевидно, что и для более высоких сингулярностей нетрудно соорудить соответствующие замены, но такие экзотические сингулярности на практике обычно не встречаются.

4.2. Релаксационные методы

Как уже было сказано, довольно часто метод «стрельбы» применить не удастся. В существенно нелинейных задачах решение обычно настолько охотно сваливается в растущий режим (при этом отнюдь не экспоненциальный или гауссов), что даже просто довести траекторию до точки сшивания не удастся. Например, в большинстве задач на поиск статических решений в нелинейных теориях поля метод «стрельбы» неприменим. Приходится прибегать к релаксационным методам.

Идея релаксационных методов довольно тривиальна. Мы отказываемся от точного решения системы ОДУ $N \times N$ с помощью тех или иных методов достаточно высокого порядка. Напротив, на нашем отрезке $[t_{start}, t_{end}]$ мы вводим сетку t_m , количество узлов в которой равно $(L + 1)$. При этом шаг обычно выбирают постоянным: $h = (t_{start} - t_{end})/L$. Тогда $t_m = t_{start} + mh$, $m = 0, \dots, L$. На этой сетке мы изображаем нашу систему ОДУ, пользуясь схемой второго порядка:

$$\frac{\vec{x}(t_{m+1}) - \vec{x}(t_m)}{h} = \frac{1}{2}\vec{f}(x(t_{m+1})) + \frac{1}{2}\vec{f}(x(t_m)), \quad m = 0 \dots (L - 1).$$

Это система $L \times N$ уравнений.⁷ Введем единое обозначение для всех этих $L \times N$ уравнений:

$$C_k^{(m)}(\vec{x}(t_m), \vec{x}(t_{m+1})) \equiv \frac{\vec{x}(t_{m+1}) - \vec{x}(t_m)}{h} - \frac{1}{2}\vec{f}(x(t_{m+1})) - \frac{1}{2}\vec{f}(x(t_m)) = 0.$$

Кроме того, мы располагаем граничными условиями на правом и левом концах:

$$\mathcal{L}_k(\vec{x}(t_0)) = 0, \quad k = 1, \dots, M,$$

$$R_k(\vec{x}(t_L)) = 0, \quad k = 1, \dots, N - M.$$

⁷Казалось бы, с равным успехом можно было бы написать

$$[\vec{x}(t_{m+1}) - \vec{x}(t_m)]/h = \vec{f}([\vec{x}(t_{m+1}) + \vec{x}(t_m)]/2),$$

но, как показывает опыт, лучше этого не делать.

Это еще N уравнений. Итак, мы имеем систему $(L+1) \times N$ уравнений на $(L+1) \times N$ неизвестных $x_i(t_k)$, каковую и предлагается решить.

В дальнейшем величины $x_i(t_k)$ будем для краткости обозначать $x_i^{(k)}$, соответственно, вместо $\vec{x}(t_k)$ будем писать $\vec{x}^{(k)}$.

4.2.1. Метод Ньютона

Как мы знаем, основным методом решения систем нелинейных уравнений является метод Ньютона (см. часть I, раздел «Многомерные корни»). На первый взгляд, применить этот метод к нашей задаче невозможно: в методе Ньютона на каждом шаге необходимо решать систему линейных уравнений, причем размер соответствующей матрицы в нашей задаче будет $[(L+1)N] \times [(L+1)N]$, а это явно нереально для любого разумного L и N . К счастью, большая часть этой грандиозной матрицы состоит из нулей, память под которые отводить не нужно. Более того, в процессе решения СЛУ методом триангуляции (см. часть I, раздел «Системы линейных уравнений») эти нули так и останутся нулями, так что в действительности количество чисел, которое надо запоминать, пропорционально не $N^2 L^2$, а $N^2 L$.

Итак, берем некоторое начальное приближение для $x_i^{(m)}$ (здесь $i = 1 \dots N$, $k = 0 \dots L$). Заметим, что это начальное приближение, вообще говоря, не обязано удовлетворять не только разностным уравнениям C_k^m , но даже и граничным условиям $\mathcal{L}_k$ и $\mathcal{R}_k$. Даем величинам $x_i^{(m)}$ приращения $\delta x_i^{(m)}$ и линеаризуем нашу систему уравнений относительно этих приращений:

$$\mathcal{L}_k(\vec{x}^{(0)}) + \partial_{x_i^{(0)}} \mathcal{L}_k(\vec{x}^{(0)}) \delta x_i^{(0)} = 0,$$

$$C_k^{(0)}(\vec{x}^{(0)}, \vec{x}^{(1)}) + \partial_{x_i^{(0)}} C_k^{(0)}(\vec{x}^{(0)}, \vec{x}^{(1)}) \delta x_i^{(0)} + \\ + \partial_{x_i^{(1)}} C_k^{(0)}(\vec{x}^{(0)}, \vec{x}^{(1)}) \delta x_i^{(1)} = 0,$$

$$C_k^{(1)}(\vec{x}^{(1)}, \vec{x}^{(2)}) + \partial_{x_i^{(1)}} C_k^{(1)}(\vec{x}^{(1)}, \vec{x}^{(2)}) \delta x_i^{(1)} + \\ \dots + \partial_{x_i^{(2)}} C_k^{(1)}(\vec{x}^{(1)}, \vec{x}^{(2)}) \delta x_i^{(2)} = 0,$$

$$C_k^{(L-1)}(\vec{x}^{(L-1)}, \vec{x}^{(L)}) + \partial_{x_i^{(L-1)}} C_k^{(L-1)}(\vec{x}^{(L-1)}, \vec{x}^{(L)}) \delta x_i^{(L-1)} + \\ + \partial_{x_i^{(L)}} C_k^{(L-1)}(\vec{x}^{(L-1)}, \vec{x}^{(L)}) \delta x_i^{(L)} = 0,$$

$$\mathcal{R}_k(\vec{x}^{(L)}) + \partial_{x_i^{(L)}} \mathcal{R}_k(\vec{x}^{(L)}) \delta x_i^{(L)} = 0.$$

Как нетрудно понять, матрица нашей СЛУ будет иметь следующую структуру:

$\partial \mathcal{L} / \partial x^{(0)}$	0		
$\partial \mathcal{C}^{(0)} / \partial x^{(0)}$	$\partial \mathcal{C}^{(0)} / \partial x^{(1)}$		
0	$\partial \mathcal{C}^{(1)} / \partial x^{(1)}$	$\partial \mathcal{C}^{(1)} / \partial x^{(2)}$	
		$\partial \mathcal{C}^{(2)} / \partial x^{(2)}$	$\partial \mathcal{C}^{(2)} / \partial x^{(3)}$
			$\partial \mathcal{R} / \partial x^{(3)}$

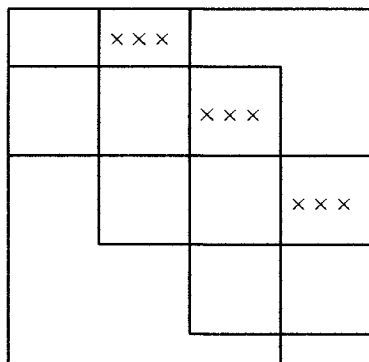
Приведенная картинка соответствует решетке с четырьмя узлами, причем $\partial \mathcal{L} / \partial x^{(0)}$ означает прямоугольную матрицу $\partial_{x_i^{(0)}} \mathcal{L}_k(\vec{x}^{(0)})$ размера $M \times N$ (поскольку $i = 1 \dots N$, $k = 1 \dots M$); $\partial C^{(0)} / \partial x^{(0)}$ означает квадратную матрицу $\partial_{x_i^{(0)}} C_k^{(0)}(\vec{x}^{(0)}, \vec{x}^{(1)})$ размера $N \times N$ и т. д. Итак, нам надо запомнить L блоков размером $2N \times N$, блок размером $M \times N$ и блок размером $(N - M) \times N$.

При триангуляции возможны два альтернативных рецепта.

Если мы стараемся минимизировать память, необходимую при решении задачи, то в стандартной процедуре возникнут некоторые ограничения. Действительно, уже на самом первом этапе, когда мы ищем максимальный по абсолютной величине элемент в первом столбце, мы должны ограничиться первыми M строчками, хотя на самом деле в первом столбце $M + N$ ненулевых элементов. Если максимальный элемент окажется на s -й строчке, причем $s > M$ (т. е. максимальный элемент возникнет из уравнения $C^{(0)}$), то мы все равно не сможем поменять местами первую и s -ую строчки. У s -й строчки есть «лишние» элементы с номерами $(N + 1) \dots 2N$, и если ее переставить на место первой строки, то эти лишние элементы некуда будет

девать — у первой строки память отведена только под элементы с номерами $1 \dots N$. Таким образом, работая с k -ым столбцом, мы ищем максимальный элемент не по всем его строкам, а только по тем, у которых длина (точнее, номер максимального ненулевого элемента) не больше, чем у k -й строки. Это единственная модификация, которую придется внести в стандартную процедуру. И довольно ясно, что эта модификация не улучшит надежность и точность метода триангуляции, зато позволит минимизировать необходимую память.

Если мы стараемся повысить надежность метода, то можно реализовать стандартную триангуляцию без всяких ограничений. Для этого придется отвести память под дополнительные блоки нашей матрицы:



Символом « $x \ x \ x$ » обозначены дополнительные блоки. Добавка к памяти составляет $(L-1)$ блок размером $N \times N$ и один блок размером $M \times N$.

Выбор между этими двумя рецептами определяется свойствами конкретной задачи. Полезно еще раз напомнить, что в методе Ньютона необходимо ограничивать смещение, т. е. длину вектора $\delta x_i^{(m)}$.

Маленькое техническое замечание. Раз мы запоминаем матрицу не целиком, а блоками, то при триангуляции возникнут чисто технические трудности, связанные с пересчетом индексов. Как именно решать эту проблему — дело вкуса. Авторы предпочитают рецепт, основанный на легкой модификации стандартной процедуры триангуляции. Мы делаем вид, что огромная матрица $N(L+1) \times N(L+1)$ существует в памяти, что обратиться можно к каждому ее элементу без всякого пересчета индекса, но фактически обращаемся только к элементам, входящим в ненулевые (рабочие) блоки. Соответственно, память отводится тоже только под эти рабочие элементы.

Прежде всего, составляем две таблицы (два одномерных целых массива `str_low` и `str_upp`) с начальными и конечными индексами для каждой строки. Начальный индекс — это где начинаются ненулевые элементы в данной строке, а конечный — где эти элементы кончаются:

Номер строки	Начальный индекс <code>str_low</code>	Конечный индекс <code>str_upp</code>
$j, j = 0 \dots M - 1$	0	$2 * N - 1$
$M + i * N + j,$ $i = 0 \dots L - 3,$ $j = 0 \dots N - 1$	$i * N$	$(i + 3) * N - 1$
$M + (L - 2) * N + j,$ $j = 0 \dots N - 1$	$(L - 2) * N$	$L * N - 1$
$M + (L - 1) * N + j,$ $j = 0 \dots N - M - 1$	$(L - 1) * N$	$L * N - 1$

(мы рассматриваем вариант с дополнительными блоками).

То же самое нужно сделать для столбцов:

Номер столбца	Начальный индекс <code>col_low</code>	Конечный индекс <code>col_upp</code>
$j, j = 0 \dots N - 1$	0	$M + N - 1$
$j,$ $j = N \dots 2N - 1$	0	$M + 2 * N - 1$
$i * N + j,$ $i = 2 \dots L - 2,$ $j = 0 \dots N - 1$	$(i - 2) * N + M$	$(i + 1) * N + M - 1$
$(L - 1) * N + j,$ $j = 0 \dots N - 1$	$(L - 3) * N + M$	$L * N - 1$

Осталось отвести память под рабочие кусочки матрицы. Саму матрицу мы определяем как массив указателей:

```
double * matr[N * (L+1)];
```

Под каждую i -ую строку надо отвести совсем немного памяти:

```
matr[i]=(double*)malloc((str_upp[i]-str_low[i]+1)*
                        sizeof(double));
if(matr[i]==NULL)
    {printf("No memory!\n"); exit(0);}
```

Теперь осталось передвинуть указатель так, чтобы отведенная память соответствовала индексам $[str_low[i] \dots str_upp[i]]$, а не индексам $[0 \dots (str_upp[i]-str_low[i]+1)]$. Для этого достаточно передвинуть индекс влево:

```
matr[i]=matr[i]-str_low[i];
```

Кто боится индексной арифметики, может воспользоваться синонимом:

```
matr[i]=&(matr[i][ -str_low[i]]);
```

Теперь осталось взять стандартную программу триангуляции и слегка ее модифицировать. Напомним, что в процедуре триангуляции самый внешний цикл идет по всем элементам диагонали. Пусть мы уже добрались до диагонального элемента A_{ii} . Тогда внутренний цикл по столбцам, который обыкновенно имеет вид

```
for(n=i; n<K; n++).
```

(здесь i — номер текущего диагонального элемента A_{ii} , до которого мы уже добрались, а K — размер матрицы) следует переписать в виде:

```
for(n=i; n<col_upp[i]; n++).
```

Аналогично, в самом внутреннем цикле по n -й строке

```
for(m=i; m<K; m++).
```

надо заменить K на $str_upp[i]$.

Если Вы нигде не ошибетесь (что весьма маловероятно), то программа заработает правильно.

4.2.2. Минимизация

Вообще-то, если метод Ньютона написан правильно, и если он сходится, то он сходится очень быстро. К сожалению, нередко одно из условий (или оба) не выполняется. Например, при очень большом разбросе в величине производных, входящих в матрицу СЛУ, метод Ньютона может не сойтись. В этом случае стоит попробовать метод минимизации невязки (см. часть I, раздел «Многомерные корни»).

Метод минимизации невязки может сойтись к решению, даже когда метод Ньютона не сходится. Кроме того, при минимизации невязки нет

необходимости решать огромную СЛУ и отводить память под элементы соответствующей матрицы. Все градиенты величин $\mathcal{L}$, $\mathcal{C}^{(m)}$ и $\mathcal{R}$ можно вычислять по ходу дела.

Иногда имеет смысл минимизировать не невязку. Если система ОДУ (система уравнений движения) получается в результате поиска экстремума того или иного функционала, то этот функционал и следует минимизировать. В частности, при поиске статических решений в одномерных нелинейных теориях поля мы ищем минимум гамильтониана поля или минимум эффективного потенциала (эффективный потенциал для бозонных полей создают другие бозонные поля или фермионы).

В этом случае разумнее не изображать систему дифференциальных уравнений на решетке (что всегда связано с некоторым произволом), а изначально рассматривать решеточную задачу. Эта решеточная задача зачастую имеет непосредственный физический смысл, например, описывает ту или иную твердотельную систему. Решеточный гамильтониан, который ее описывает, приводит к вполне однозначной системе разностных уравнений для статического решения.

Поиск минимума решеточного гамильтониана написать гораздо проще, чем поиск минимума невязки в соответствующей системе разностных уравнений. Действительно, при поиске минимума гамильтониана нам достаточно вычислять сам гамильтониан и его первые производные («силы», действующие на решеточные степени свободы). Как только мы найдем минимум, силы автоматически занулятся, и мы получим искомое статическое решение. Между тем условие обращения всех сил в ноль дает в точности ту систему уравнений, которую мы решали в предыдущем разделе. Если мы станем решать эту систему уравнений методом минимизации невязки, то нам придется вычислять еще и градиенты всех сил, т. е. вторые производные гамильтониана. В нелинейных задачах выражения для вторых производных, как правило, гораздо более громоздкие, чем для первых.

Таким образом, есть по крайней мере две причины, по которым иногда стоит отказаться от метода Ньютона в пользу более медленного метода минимизации:

- Когда метод Ньютона не сходится, поиск минимума гораздо надежнее. Например, для положительно определенного решеточного гамильтониана динамический метод обязательно сойдется, и, скорее всего, к основному состоянию.
- Когда метод Ньютона никак не напишется правильно. Написать минимизацию гамильтониана гораздо проще. Поэтому лучше за 20 минут написать простую и правильную программу минимизации, которая бу-

дет считать 2 минуты, чем 2 дня искать ошибки в программе с методом Ньютона, которая будет считать 20 секунд.

Искать многомерный минимум можно разными методами (см. часть I, раздел «Многомерные минимумы»), но для данной задачи рекомендация авторов однозначна — следует использовать динамический метод.

Здесь необходимо маленькое отступление о соотношении решеточной и исходной непрерывной теории поля.

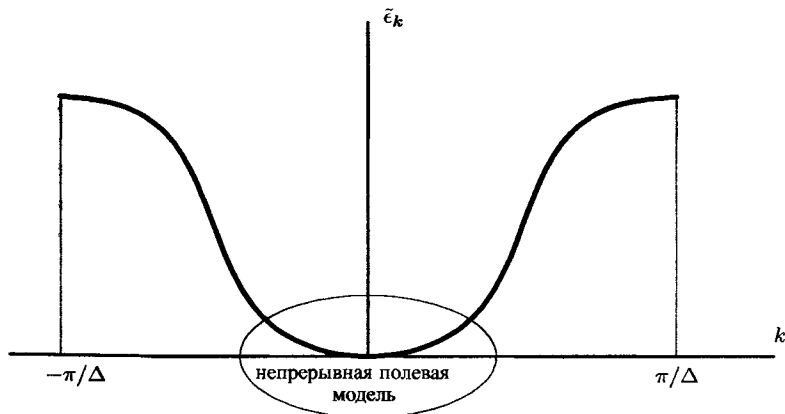
Прежде всего, как это ни странно, динамика решеточной модели зачастую богаче динамики исходной теории поля. Рассмотрим, например, безмассовое одномерное скалярное поле с гамильтонианом

$$H = \int dx \left[\frac{1}{2} (\partial_t \phi(x, t))^2 + \frac{1}{2} (\partial_x \phi(x, t))^2 \right].$$

Этой непрерывной полевой модели соответствует решеточная модель — просто система грузов и пружинок:

$$H_{lattice} = \sum_n \Delta x \left[\frac{1}{2} (\partial_t \phi_n)^2 + \frac{1}{2} \left(\frac{\phi_{n+1} - \phi_n}{\Delta x} \right)^2 \right].$$

Если мы перейдем к собственным колебаниям (т.е. к плоским волнам), то зависимость энергии от волнового числа в непрерывном случае будет $\epsilon_k = k^2/2$, а в решеточном случае будет $\tilde{\epsilon}_k = [1 - \cos(k\Delta x)]/\Delta x^2$:



В окрестности точки $k = 0$, точнее, при малом $k\Delta x$ эти выражения практически совпадают. Однако (при сколь угодно малом Δx !) косинус все равно

останется косинусом, и большая часть решеточных степеней свободы никакого отношения к полевой динамике не имеет. Уменьшение Δx никак не меняет относительного числа (доли) степеней свободы, соответствующих непрерывной полевой модели.

Может показаться, что степени свободы с волновыми числами порядка $k = \pi/(2\Delta x)$ или $k = \pi/\Delta x$ имеют слишком большую энергию и, следовательно, не могут иметь никакой нетривиальной динамики, даже в случае нелинейной полевой модели. Это совершенно не так. Рассмотрим очень простую одномерную решеточную модель:

$$H = \sum_n \frac{1}{2} (\partial_t \phi_n)^2 + \frac{1}{2} \left[\frac{\phi_{n+1} - \phi_n}{\Delta x} \right]^2 - [\alpha - \beta(\phi_{n+1} - \phi_n)] (c_{n+1}^+ c_n + c_n^+ c_{n+1}). (*)$$

Здесь ϕ_n — скалярное поле, c_n — фермионные степени свободы.⁸ Легко проверить, что в основном состоянии гармоника скалярного поля с волновым числом $k = \pi/\Delta x$ будет иметь ненулевую амплитуду. Происходит это из-за взаимодействия с фермионными степенями свободы, волновые числа которых лежат около $k = \pm\pi/(2\Delta x)$. Так что если Вы возьметесь считать на решетке динамику одномерной полевой модели

$$H = \int dx \left[\frac{1}{2} (\partial_t \phi(x))^2 + \frac{1}{2} (\partial_x \phi(x))^2 + (A + B \partial_x \phi(x)) [ic^+(x) \partial_x c(x) - ic(x) \partial_x c^+(x)] \right], (**)$$

где A и B — константы, то у Вас получатся результаты, не имеющие ни малейшего отношения к исходной непрерывной модели. Вся нетривиальная динамика решеточного гамильтониана (*) лежит в области $k = \pm\pi/(2\Delta x)$ для фермионов и $k = \pi/\Delta x$ для бозона, а исходная полевая модель — это окрестность $k = 0$.

Итак, динамика решеточной модели действительно богаче исходной полевой динамики.

Но это не значит, что не существует полевой модели, которая описывается решеточным гамильтонианом (*). Парадокс заключается в том, что не в непрерывной модели заключено бесконечно много решеточных, а наоборот, в решеточной заключено бесконечно много разных полевых моделей. Именно, окрестности $k = \pi/\Delta x$ (точно так же, как и окрестности $k = 0$) соответствует полевая модель. Чтобы ее построить, достаточно полагать

⁸Этот решеточный гамильтониан описывает вполне реальную систему — полимерную цепочку транс-полиацетилена, который является диэлектриком Пайерлса.

непрерывной функцией координаты x не величину ϕ_n , а величину $(-1)^n \phi_n$. Совершенно аналогично, для фермионов надо работать с двумя непрерывными величинами $\psi_{\pm}(x_n) = (\pm i)^n c_n$. В результате получится полевая модель, которая вполне соответствует решеточному гамильтониану (*):

$$H = \int dx \frac{1}{2} (\partial_t \phi(x))^2 + C \frac{1}{2} (\varphi)^2 + D (i\psi^+ + \partial_x \psi_+ - i\partial_x \psi^+ + \psi_+) + \\ + D (i\psi^+ - \partial_x \psi_- - i\partial_x \psi^+ - \psi_-) + E \phi (\psi^+ + \psi_- + \psi^+ - \psi_-),$$

где C , D и E — константы.⁹

Кроме того, не следует думать, что от влияния концевых точек $X_{\pm}$ всегда можно избавиться, увеличивая число узлов, т. е. удлиняя решетку. Есть задачи, в которых влияние граничных условий глобально и не зависит от длины решетки. Самый простой пример такой ситуации — это полевые модели с топологическими солитонами. Например, в стандартной ϕ^4 -модели

$$H = \int dx \left[\frac{1}{2} (\partial_t \phi(x))^2 + \frac{1}{4} (\partial_x \phi(x))^2 + \frac{1}{2} (\phi(x)^2 - 1)^2 \right]$$

при граничных условиях $\phi(X_-) = 1$, $\phi(X_+) = 1$ основным состоянием будет константа $\phi(x) = 1$, а при граничных условиях $\phi(X_-) = -1$, $\phi(X_+) = 1$ основным состоянием будет солитон $\phi(x) = \tanh(x)$.

Более нетривиальный пример — это модель с гамильтонианом (**). В этой модели при периодических граничных условиях (или при закрепленных концах) мы получим один ответ, а при свободных концах — совершенно другой. Это связано с тем, что при свободных концах возникнет конденсат в первой производной поля ϕ : $\partial_x \phi(x) = c_0 = \text{const} \neq 0$.

Итак, при численном решении задач нелинейной теории поля необходимо помнить, что полученные Вами результаты не всегда имеют отношение к исходной полевой модели. Более того, использование очень мелкого шага и очень большого числа узлов не всегда спасает ситуацию.

4.2.3. Переменный шаг

Нехватка памяти и медленный счет из-за большого числа узлов — это обычная проблема при использовании релаксационных методов. Если мы пользуемся равномерной решеткой, то уменьшение числа узлов означает увеличение шага, т. е. мы получаем менее точный ответ.

⁹Эта полевая модель есть упрощенный вариант модели Гросса-Невье, в которой происходит динамическое нарушение симметрии, т. е. ненулевая компонента скалярного поля («конденсат») появляется не из-за специально написанного потенциала, а за счет взаимодействия с фермионами.

Если мы работаем с решением типа солитона, который имеет нетривиальное поведение в некоторой ограниченной области посередине интервала $[t_1, t_2]$ (в этой области правая часть ОДУ достаточно велика), а ближе к концам выходит на константу (в этих областях правая часть ОДУ практически равна нулю), то можно уменьшить число узлов, почти не ухудшая точность. Для этого следует использовать решетку с переменным шагом. Именно, в нетривиальной области посередине интервала взять шаг помельче, а в остальных местах сделать шаг побольше.

Вообще-то при работе с неравномерными решетками надо быть очень осторожным, и стоит хорошо подумать, прежде чем их использовать. Дело в том, что они вполне могут стать источником дополнительных ошибок.

Если из физических соображений заранее известно, где именно будет нетривиальная область, то можно изначально задать фиксированную неравномерную решетку со сгущением узлов в нетривиальной области. Если априорной информации о поведении решения нет, то процедуру построения неравномерной решетки можно автоматизировать.

Именно, мы проводим в нашей системе ОДУ

$$\frac{d}{dt}\vec{x}(t) = \vec{f}(\vec{x}(t)) \quad (*)$$

замену переменных $t \rightarrow \tau$:

$$\frac{d}{d\tau}\vec{x}(\tau) = \frac{dt}{d\tau}\vec{f}(\vec{x}(\tau)) \quad (**)$$

и вводим две дополнительные переменные: $x_{N+1}(\tau) \equiv t(\tau)$ и $x_{N+2}(\tau)$. Их уравнения движения по определению имеют вид

$$\frac{dt}{d\tau} = \frac{d}{d\tau}x_{N+1}(\tau) = \frac{x_{N+2}(\tau)}{1 + A\|\vec{f}(\vec{x}(\tau))\|}, \quad \frac{d}{d\tau}x_{N+2}(\tau) = 0.$$

Здесь A — константа, а $\|\vec{f}(\vec{x}(\tau))\|$ — какая-нибудь норма, измеряющая величину правой части системы ОДУ. Константу A и норму следует подбирать, исходя из конкретной задачи.

Новое «время» τ меняется на отрезке $[\tau_1, \tau_2]$. При этом должно быть $t(\tau_1) = t_1$ и $t(\tau_2) = t_2$. Так что переменная x_{N+1} должна удовлетворять граничным условиям $x_{N+1}(\tau_1) = t_1$ и $x_{N+1}(\tau_2) = t_2$.

Если ввести равномерную сетку по τ на отрезке $[\tau_1, \tau_2]$, то сетка по t будет измельчаться как раз в нетривиальных областях, т. е. там, где велика правая часть системы ОДУ. Эффект от такого измельчения в преобразованной системе (**) виден непосредственно. Действительно, если правая часть

исходной системы ОДУ (*) $\vec{f}$ вдруг резко возрастает, то возрастает и норма $\|\vec{f}\|$. При этом резко убывает множитель $dt/d\tau$ в правой части (**), что отчасти компенсирует рост $\vec{f}$. Поэтому правая часть системы (**) не будет иметь таких резких пиков, как правая часть исходной системы (*), и мы (при данной заказанной точности) можем использовать более крупный шаг по τ , т. е. обойтись меньшим числом узлов.

Разумеется, убывание величины $dt/d\tau$ в нетривиальной области приводит к измельчению шага по t , т. е. к сгущению сетки по t в нетривиальной области.

Вторая переменная x_{N+2} есть нормировочный множитель, который позволяет функции $t(\tau)$ попасть в точку t_2 при $\tau = \tau_2$. Без такого множителя эволюция переменной x_{N+1} полностью определялась бы нормой $\|\vec{f}\|$ и мы, скорее всего, не смогли бы удовлетворить граничному условию $x_{N+1}(\tau_2) = t_2$.

Метод, конечно, очень изящный, но пользоваться им надо с осторожностью. Во-первых, как уже было сказано, сам по себе переменный шаг иногда может увеличивать погрешности в ответе. Во-вторых, этот метод иногда сходится очень медленно (или вовсе не сходится). Дело в том, что дополнительные переменные x_{N+1} и x_{N+2} не только очень сильно влияют на поведение исходных переменных $x_1 \dots x_N$, но испытывают на себе и обратное влияние, поэтому есть опасность, что они «раскачают» весь релаксационный процесс.

5. «Жесткие» системы

Представим себе, что мы численно решаем следующую систему ОДУ:

$$\dot{x}_1 = -x_1, \quad \dot{x}_2 = -10000 x_2.$$

Точное решение имеет вид

$$x_1(t) = x_1(0) \exp(-t), \quad x_2(t) = x_2(0) \exp(-10000 t).$$

Довольно ясно, что любой алгоритм с адаптивным изменением шага немедленно установит шаг h , который заведомо будет гораздо меньше 10^{-4} , например, может получиться шаг h порядка 10^{-6} . С точки зрения эволюции переменной x_1 такой шаг является несуразно маленьким, но он абсолютно необходим для правильного описания эволюции переменной x_2 .

Между тем, бывают задачи, в которых динамика «быстрой» переменной x_2 нам совершенно неинтересна. Нам интересна только динамика

«медленной» переменной x_1 . Даже в нелинейной задаче, где существует взаимное влияние «медленных» и «быстрых» переменных, для описания динамики «медленной» переменной x_1 нам не нужно знать точный закон эволюции переменной x_2 . Нам достаточно знать, что эта переменная очень быстро (за время порядка 10^{-4}) обращается в ноль и в дальнейшем нулем и остается. Более того, если время жизни переменной x_2 увеличится в 10 раз (т.е. $\dot{x}_2 = -1000 x_2$), то даже в нелинейной задаче на эволюцию x_1 это почти не повлияет.

Поэтому возникает идея использовать такой шаг по времени h , который позволит правильно описать динамику «медленной» переменной x_1 , т.е. шаг порядка 0.01. Этот шаг заведомо не позволит правильно отследить динамику «быстрой» переменной x_2 , но с этим мы готовы смириться. Если при выбранном шаге h переменная x_2 будет вести себя как $\exp(-\alpha t)$ с каким-нибудь, пусть и совершенно неправильным, параметром α , этого будет достаточно для приблизительно правильного описания динамики x_1 .

К сожалению, все отнюдь не так просто.

Применим схему Рунге-Кутты первого порядка точности. Тогда для «медленной» переменной мы получим $x_1(h) = x_1(0)[1 - h]$ вместо $x_1(h) = x_1(0) \exp(-h)$, что не так уж неправильно при $h = 0.01$. Однако для быстрой переменной будет $x_2(h) = x_2(0)[1 - 10000 h]$, что при $h = 0.01$ дает $x_2(h) = -99 x_2(0)$. Нетрудно сообразить, что $x_2(2h) = (-99)^2 x_2(0)$, так что вместо экспоненциального убывания мы получили экспоненциальный рост.

При этом увеличение порядка схемы с одного до двух (или, скажем, до пяти) отнюдь не улучшит ситуацию, поскольку любая схема может работать, только если правая часть уравнения для x_2 , умноженная на шаг, является достаточно маленькой величиной по сравнению с x_2 . У нас в любом случае получится $[10000 h] x_2$, что при $h = 0.01$ означает $100 x_2$.

Проверим эти утверждения непосредственно. Применяя схему Рунге-Кутты второго порядка (которая, напомним, используется в интерполяционных методах), получим

$$\begin{aligned} x_2(h) &= x_2(0) - 10000 h x_2(h/2) = \\ &= x_2(0) - 10000 h x_2(0) [1 - 10000 (h/2)] = \\ &= x_2(0) [1 - (10000 h) + (10000 h)^2/2]. \end{aligned}$$

Хотя мы, как и следовало ожидать, получили два первых слагаемых степенного ряда для экспоненты $\exp(-10000 h)$, проку от этого мало, поскольку при $h = 0.01$ число 4901 явно не имеет ничего общего с $\exp(-100)$.

Выход заключается в следующем: вместо явных схем надлежит применять неявные. Неявная схема Рунге-Кутты первого порядка имеет вид:

$$\vec{x}(t+h) = \vec{x}(t) + h\vec{f}(\vec{x}(t+h)).$$

Это не явное выражение для $\vec{x}(t+h)$, а система уравнений, причем, вообще говоря, нелинейная. Эту систему еще надо решить.

Для игрушечного примера, который мы рассматриваем, получится линейное уравнение:

$$x_2(t+h) = x_2(0) - 10000 h x_2(t+h),$$

откуда

$$x_2(t+h) = x_2(0)/(1 + 10000 h).$$

Разумеется, при $h = 0.01$ число $1/101$ не слишком похоже на $\exp(-100)$, но величина x_2 , по крайней мере, экспоненциально убывает. Разумеется, время жизни будет совершенно неправильное, но это и неважно: за, приблизительно, 7 шагов x_2 умрет до $x_2(0) \cdot 10^{-14}$, т.е. практически до машинного нуля, а x_1 за это время почти не сдвинется.

Утверждение о том, что неявные схемы предотвращают экспоненциальную неустойчивость, возникающую при неправильном (очень большом) шаге по времени, является достаточно общим. Словесное объяснение (но не доказательство) этого факта довольно простое: если в схеме возникает экспоненциальный рост из-за большого шага, то он, очевидно, возникает при движении в обе стороны по времени. Выполняя один шаг явной схемы, мы движемся в сторону возрастания времени и, следовательно, получаем экспоненциальный рост по времени, т.е. неустойчивость. Неявная схема приводит к тому, что в пределах данного шага мы, в сущности, движемся по времени в обратную сторону (мы используем не текущую, а будущую производную). Из-за этого мы получим экспоненциальный рост, но уже в сторону убывания времени, а это означает экспоненциальное убывание с ростом времени, т.е. устойчивость. Мы еще встретимся с неявными схемами при решении параболических дифференциальных уравнений в частных производных.

Основная проблема неявных схем заключается в необходимости решать нелинейную систему уравнений. Существуют разные рекомендации по этому поводу.

Иногда утверждается, что лучше методом Ньютона решить нелинейную систему уравнений до конца, т.е. сойтись к решению. Обычно на это

уходит не более 5 итераций, поскольку корень $\vec{x}(t+h)$ лежит не так уж далеко от начального приближения к корню $\vec{x}(t)$.

Как нам кажется, в большинстве случаев достаточно ограничиться линейным приближением, т.е. выполнить только первую итерацию метода Ньютона. Только в исключительно неустойчивых задачах имеет смысл попробовать реализовать метод Ньютона до конца, т.е. действительно найти решение исходной нелинейной системы уравнений.

Линеаризованная неявная схема позволяет ограничиться решением СЛУ. Например, рассмотрим неявную схему второго порядка:

$$\vec{x}(t+h) = \vec{x}(t) + h\vec{f}\left(\frac{\vec{x}(t+h) + \vec{x}(t)}{2}\right).$$

Заметим, кстати, что эта схема второго порядка является односточечной. Линеаризация правой части относительно смещения $\delta\vec{x} = \vec{x}(t+h) - \vec{x}(t)$ даст

$$x_i(t+h) = x_i(t) + hf_i(\vec{x}(t)) + h\partial_{x_j}f_i(\vec{x}(t))\delta x_j/2,$$

откуда

$$\left(\delta_{ij} - (h/2)\partial_{x_j}f_i(\vec{x}(t))\right)\delta x_j = hf_i(\vec{x}(t)).$$

Так что в линейном приближении достаточно один раз решить СЛУ, чтобы сделать один шаг по времени $t \rightarrow t+h$. Это в точности соответствует первой итерации метода Ньютона, если в качестве начального приближения к корню $\vec{x}(t+h)$ брать $\vec{x}(t)$.

Еще одна проблема заключается в том, как контролировать точность счета. Мы по определению не интересуемся точной эволюцией «быстрых» переменных, так что точность счета определяется точностью «медленных» переменных. Но если бы мы могли одни переменные отцепить от других, не нужно было бы применять неявные схемы. Поэтому вопрос в том, как именно вектор погрешностей $\vec{E}$ превращать в одно число ϵ , характеризующее погрешность (см. раздел «Методы Рунге-Кутты»), для «жестких» систем становится особенно запутанным. Здесь нас постоянно подстерегают две опасности — зависеть точность по «медленным» переменным из-за переоценки погрешности, вызванной в основном «быстрыми» переменными; либо, наоборот, получить совершенно неправильный ответ для «медленных» переменных, считая, что большая погрешность вызвана только «быстрыми» переменными.

Итак, при решении «жестких систем» дифференциальных уравнений придется использовать неявные схемы. Существуют неявные мето-

ды Рунге-Кутта, неявные интерполяционные методы и неявные методы «предсказание-коррекция». По причинам, изложенным в разделе «Задача Коши для ОДУ», мы ограничимся двумя первыми группами методов.

Вообще, наша основная рекомендация такова: если есть хоть малейшая возможность выделить нефизические «быстрые» степени свободы, выкинуть их и оставить только интересующие нас «медленные» степени свободы, то, как бы громоздко ни выглядели полученные выражения, лучше это сделать — это окупится как в отношении точности результатов, так и в отношении скорости счета.

Приведем два простейших, но довольно наглядных примера.

Допустим, мы рассматриваем динамику материальной точки в двумерном пространстве. Точка движется в потенциале

$$V(x, y) = C(x^2 + y^2 - 1)^2 + x^2/2,$$

где $C = 10^6$. Очевидно, что если энергия материальной точки порядка единицы, то она движется по окружности $x^2 + y^2 = 1$, причем возможные отклонения составляют не более чем $\delta x = 1/\sqrt{C} = 10^{-3}$. Поэтому напрашивается замена переменных $x = r \cos \varphi$, $y = r \sin \varphi$. При этом угол φ является «медленной» переменной, а радиальная переменная r — «быстрой».

Следовательно, мы фактически можем решать одномерную задачу с потенциалом

$$V(\varphi) = (1/2) \cos^2 \varphi.$$

Более того, если нам совершенно необходимо учитывать еще и малые колебания по радиусу, то можно определить новую переменную ρ :

$$r = 1 + \rho/\sqrt{C},$$

тогда

$$V(\rho, \varphi) \approx 4\rho^2 + (1/2) \cos^2 \varphi.$$

Эта задача может быть решена обычными (явными) методами.

Если мы станем решать эту задачу «в лоб», т. е. в терминах исходных переменных x и y , то нам придется использовать неявные схемы. При этом описание динамики «медленной» переменной φ будет вполне удовлетворительным, а вот описание динамики «быстрой» переменной r будет неправильным. Точнее, оно будет правильным качественно — r будет колебаться в окрестности $r = 1$, но эти колебания будут совершенно непохожи на правильные колебания r .

Следовательно, в таких задачах неявные схемы применять можно, но мы получим гораздо более качественное описание системы, если расцепим переменные и перемасштабируем их.

Другой пример: одномерное движение материальной точки в потенциале

$$V(x) = C x \theta(x),$$

где $C = 10^6$. Пусть частица с энергией порядка единицы движется в направлении потенциального барьера из области $x \rightarrow -\infty$. Довольно ясно, что в область положительных x она проникнет не дальше, чем на $\delta x = 1/C$, и проведет там времени не больше, чем $\delta t = 1/C$, после чего упруго отразится от барьера. Поэтому можно наложить «граничное условие» при $x = 0$: если при движении частицы оказалось, что $x = 0$ и $\dot{x} = v > 0$, то следует немедленно положить $\dot{x} = -v$, что соответствует абсолютно упругому удару.

Если попытаться решать эту задачу «в лоб», то нас ждет неприятный сюрприз. Только при очень существенном уменьшении шага в окрестности точки $x = 0$ мы получим правильное значение $\dot{x} = -v$ для скорости после удара о барьер. Объяснение очень простое: при ударе «медленная» переменная x превращается в «быструю», а после удара снова превращается в «медленную». Поэтому, чтобы получить правильную скорость после удара, мы обязаны найти правильную динамику «быстрой переменной».

Следовательно, в такого рода задачах разумнее накладывать «граничные» условия, а не надеяться на неявные схемы.

Итак, приведенными в этом разделе алгоритмами следует пользоваться только от отчаяния.

5.1. Неявные схемы Рунге-Кутта

Изложенный здесь алгоритм есть неявная схема Рунге-Кутта четвертого порядка. В нее, как обычно, встроена оценка погрешности, которая равна разнице ответов четвертого и третьего порядков точности, вычисленных по использованным точкам. Иными словами, по сравнению с рекомендованным выше явным методом Рунге-Кутта пятого порядка, все порядки в точности на единицу меньше. Напомним, что в методе Рунге-Кутта пятого порядка используется шесть точек. В неявной схеме точек будет гораздо меньше, поскольку для неявных схем связь порядка точности с количеством использованных точек не такая, как для явных схем. Например, как мы уже видели, одноточечная неявная схема может иметь второй порядок точности.

В нашей неявной схеме четвертого порядка мы обойдемся всего-навсего тремя точками:

$$\vec{f}_i = \vec{f}(\vec{x} + \delta\vec{x}_i),$$

где

$$\delta\vec{x}_1 = 0,$$

$$\delta\vec{x}_2 = h\hat{A}^{-1}\vec{f}_1,$$

$$\delta\vec{x}_3 = h[(3/25)\hat{A}^{-1}\vec{f}_2 + (24/25)\hat{A}^{-1}\vec{f}_1 + (-12/25)\hat{A}^{-2}\vec{f}_1],$$

причем

$$A_{ij} = \left[\delta_{ij} - \frac{h}{2} \partial_{x_j} f_i(\vec{x}) \right].$$

Смещение $\delta\vec{x}$ и погрешность $\vec{E} = \delta\vec{x} - \delta\vec{x}'$ мы ищем в виде:

$$\begin{aligned} \delta\vec{x} = h & \left(\frac{19}{18}\hat{A}^{-1} - \frac{43}{108}\hat{A}^{-2} - \frac{11}{12}\hat{A}^{-3} + \frac{5}{9}\hat{A}^{-4} \right) \vec{f}_1 + \\ & + h \left(\frac{1}{4}\hat{A}^{-1} + \frac{1}{72}\hat{A}^{-2} - \frac{5}{36}\hat{A}^{-3} \right) \vec{f}_2 + h \left(\frac{25}{36}\hat{A}^{-1} - \frac{25}{216}\hat{A}^{-2} \right) \vec{f}_3, \end{aligned}$$

$$\begin{aligned} \vec{E} = h & \left(\frac{17}{108}\hat{A}^{-1} - \frac{35}{54}\hat{A}^{-2} - \frac{13}{36}\hat{A}^{-3} + \frac{5}{9}\hat{A}^{-4} \right) \vec{f}_1 + \\ & + h \left(\frac{7}{72}\hat{A}^{-1} - \frac{1}{8}\hat{A}^{-2} - \frac{5}{36}\hat{A}^{-3} \right) \vec{f}_2 + h \left(\frac{125}{216}\hat{A}^{-1} - \frac{25}{216}\hat{A}^{-2} \right) \vec{f}_3. \end{aligned}$$

На первый взгляд, следует сначала вычислить матрицу $\hat{A}^{-1}$, а затем применять ее нужное количество раз ко всем f_i . Однако, как мы знаем (см. часть I, раздел «Системы линейных уравнений»), это не есть правильный способ решения СЛУ «впрок». Правильный способ решения СЛУ с даниой матрицей $\hat{A}$ раз и навсегда (для произвольной правой части) — это LU -разложение матрицы $\hat{A}$. При этом матрица $\hat{A}$ раскладывается на левую нижнюю треугольную матрицу L и правую верхнюю треугольную матрицу U . После этого решение любой СЛУ с матрицей $\hat{A}$ строится с помощью прямой подстановки (для матрицы L) и затем обратной подстановки (для матрицы U). Каждая из подстановок требует N^2 операций.

В нашем случае имеет смысл скомбинировать правые части всех СЛУ так, чтобы количество применений матрицы $\hat{A}^{-1}$ (точнее, количество СЛУ, которые надо решить) было минимальным. Для этого достаточно ввести

промежуточные переменные $\vec{y}_1, \vec{y}_2, \vec{y}_3$ и $\vec{y}_4$. В результате мы получим следующий рецепт для одного шага по времени $t \rightarrow t + h$.

Алгоритм:

Для текущей точки $\vec{x}(t)$ вычисляем $\vec{f}_1 = \vec{f}(\vec{x})$. Кроме того, вычисляем $A_{ij} = \delta_{ij} - (h/2)\partial_{x_j} f_i(\vec{x})$.

Выполняем LU -разложение матрицы $\hat{A}$. (Это делается один раз и навсегда для данного шага по времени $t \rightarrow t + h$).

С помощью LU -разложения вычисляем $y_1 = \hat{A}^{-1} f_1$.

Вычисляем $\delta \vec{x}_2 = h y_1$. Вычисляем $\vec{f}_2 = \vec{f}(\vec{x} + \delta \vec{x}_2)$. С помощью LU -разложения вычисляем $y_2 = \hat{A}^{-1}(f_2 - 4y_1)$.

Вычисляем $\delta \vec{x}_3 = h[(3/25)y_2 + (24/25)y_1]$. Вычисляем $\vec{f}_3 = \vec{f}(\vec{x} + \delta \vec{x}_3)$. С помощью LU -разложения вычисляем

$$y_3 = \hat{A}^{-1}(f_3 + (186/25)y_1 + (6/5)y_2).$$

С помощью LU -разложения вычисляем

$$y_4 = \hat{A}^{-1}(f_3 - (56/25)y_1 - (27/25)y_2 - (1/5)y_3).$$

Вычисляем значение $\vec{x}(t + h)$:

$$\begin{aligned} \vec{x}(t + h) = \vec{x}(t) + (19/18)\vec{y}_1 + \\ + (1/4)\vec{y}_2 + (25/216)\vec{y}_3 + (125/216)\vec{y}_4. \end{aligned}$$

Вычисляем погрешность $\vec{E}$:

$$\vec{E} = (17/108)\vec{y}_1 + (7/72)\vec{y}_2 + 0 + (125/216)\vec{y}_4.$$

Превращаем вектор $\vec{E}$ в одно число ϵ , характеризующее погрешность на данном шаге.

Эта схема приводит к довольно забавному поведению «быстрых» переменных. Поведение оказывается совершенно одинаковым как в случае релаксационного уравнения $\dot{x} = -Cx$, так и в случае осцилляторного уравнения $\ddot{x} = -Cx$. Именно, при $Ch \gg 1$ «быстрые» переменные экспоненциально убывают, причем скорость убывания совершенно не зависит от величины C : $x(t + h) = x(t)/3$. Так что устойчивость гарантирована, но надеяться на правильное описание эволюции «быстрых» переменных не придется.

Полный алгоритм решения системы ОДУ с адаптивным изменением шага строится на основе этого рецепта точно таким же образом, как строился полный алгоритм в обычном (явном) методе Рунге-Кутты пятого порядка. Единственная модификация заключается в том, что степени « $1/4$ » и « $1/5$ », соответствующие пятому порядку точности явной схемы, надлежит

заменить на степени « $1/3$ » и « $1/4$ », соответствующие четвертому порядку точности неявной схемы.

Необходимо также напомнить, что для «жесткой» системы ОДУ превратить вектор $\vec{E}$ в одно число ϵ , оценивающее погрешность, гораздо сложнее, чем для обычной системы ОДУ.

5.2. Неявные интерполяционные схемы

Алгоритм неявных интерполяционных методов дословно повторяет алгоритм обычных (явных) интерполяционных методов. Единственное отличие заключается в том, что мы не используем обычную (явную) схему Рунге-Кутты второго порядка:

$$\begin{aligned}\vec{f}_1 &= \vec{f}(\vec{x}(t) + \delta\vec{x}_1), \quad \delta\vec{x}_1 = 0; \\ \vec{f}_2 &= \vec{f}(\vec{x}(t) + \delta\vec{x}_2), \quad \delta\vec{x}_2 = h/2\vec{f}_1; \\ \vec{x}(t+h) &= \vec{x}(t) + h\vec{f}_2.\end{aligned}$$

Мы заменяем ее на неявную схему Рунге-Кутты второго порядка:

$$\vec{x}(t+h) = \vec{x}(t) + h\vec{f}([\vec{x}(t+h) + \vec{x}(t)]/2).$$

Напомним, что для этой схемы можно использовать два разных рецепта:

- искать точное решение этой системы нелинейных уравнений методом Ньютона;
- ограничиться линеаризованным уравнением, т.е. сделать только первый шаг метода Ньютона:

$$\left(\delta_{ij} - (h/2)\partial_{x_j} f_i(\vec{x}(t))\right)(x_j(t+h) - x_j(t)) = hf_i(\vec{x}(t)).$$

В любом случае придется решать СЛУ. В большинстве случаев можно обойтись вторым вариантом, но в очень неустойчивых задачах придется прибегнуть к первому (кстати, при этом не придется сильно менять текст программы).

Напомним, что существуют варианты интерполяционного метода с фиксированным шагом H и переменным порядком m , с фиксированным порядком m и переменным шагом H и, наконец, с переменными m и H . Здесь мы приведем только простейший рецепт с фиксированным H . Мы надеемся, что при необходимости остальные рецепты читатели построят самостоятельно.

Алгоритм:

Выбираем постоянный шаг H по времени (как правило следует брать H порядка 0.2). Движемся по траектории с этим шагом. Каждый отдельный шаг реализуется так:

Для $k = 1, 2, \dots$ выполняем следующее:

Для очередного N_k из набора $\{4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128\}$ вычисляем шаг $h_k = H/N_k$.

С помощью данного шага h_k находим $\vec{x}^{(k)}(t+H)$:

$$x_j(t_{n+1}) = x_j(t_n) + \left(\delta_{ij} - \frac{h}{2} \partial_{x_j} f_i(\vec{x}(t_n)) \right)^{-1} \left[h f_i(\vec{x}(t_n)) \right],$$

здесь $t_n = t + nh_k$, $n = 0 \dots N_k - 1$; при этом $\vec{x}^{(k)}(t+H) \equiv \vec{x}(t + N_k h_k)$. Разумеется, здесь не следует вычислять обратную матрицу, надо попросту решать СЛУ.

Для каждого x_i по k точкам:

$$(X_1 = h_1^2, Y_1 = x_i^{(1)}(t+H)),$$

$$(X_2 = h_2^2, Y_2 = x_i^{(2)}(t+H)),$$

...

$$(X_k = h_k^2, Y_k = x_i^{(k)}(t+H)),$$

выполняем полиномиальную интерполяцию зависимости $Y(X)$ в точку $X = 0$. Результат интерполяции, полученный на k -ом этапе, обозначаем $\tilde{x}_i^{(k)}(t+H)$.

Оцениваем погрешность $\vec{E}^{(k)}$ как

$$E_i^{(k)} = \tilde{x}_i^{(k)}(t+H) - \tilde{x}_i^{(k-1)}(t+H).$$

Превращаем вектор $\vec{E}^{(k)}$ в число $\epsilon^{(k)}$, характеризующее погрешность. Если $\epsilon^{(k)}$ меньше заказанного ϵ_0 , то полагаем $x_i(t+H) \equiv \tilde{x}_i^{(k)}(t+H)$ и завершаем данный шаг $t \rightarrow t+H$.

Первое неприятное отличие от явного метода заключается в том, что каждый маленький шагик на h_k (большой шаг H складывается из N_k штук таких шажков) требует решения СЛУ.

Второе неприятное отличие от явного метода заключается в том, что здесь гораздо сложнее превратить вектор $\vec{E}^{(k)}$ в одно число $\epsilon^{(k)}$, оценивающее погрешность. К сожалению, рецепт этого превращения следует под-

бирать в зависимости от конкретной задачи, т. е. от конкретных свойств «быстрых» и «медленных» переменных.

На первый взгляд, этот алгоритм приведет не просто к количественным ошибкам в описании динамики «быстрых» переменных, а к ошибкам качественным. Действительно, если уравнение для «быстрой» переменной имеет вид $\dot{x} = -Cx$, то при $Ch \gg 1$ неявная схема Рунге-Кутты второго порядка вместо экспоненциального убывания дает $x(t+h) \approx -x(t)$, т. е. $|x(t+H)| \approx |x(t)|$. К счастью, последующая интерполяция исправляет ситуацию. Интерполяционный алгоритм «замечает», что по мере уменьшения h_k абсолютная величина $x^{(k)}(t+H)$ также уменьшается. Причем это уменьшение тем заметнее, чем меньше h_k . В результате окончательный ответ $\tilde{x}^{(k)}(t+H)$ будет заметно меньше по абсолютной величине, чем $x(t)$.

6. Дифференциальные уравнения в частных производных

Мы будем необычайно лаконичны при изложении этой темы — ограничимся лишь простейшими и универсальными рецептами. Оправданием нам может служить то обстоятельство, что, как ни странно, численное решение дифференциальных уравнений в частных производных (ДУЧП) довольно редко встречается в практической работе теоретика, занимающегося квантовой теорией поля или теорией гравитации. Во-первых, в этих областях обычно приносят пользу только аналитические решения. Даже приближенное аналитическое решение лучше, чем точное численное, поскольку без явного аналитического выражения очень трудно двигаться дальше. Во-вторых, обычно работают с симметричными решениями. Опять-таки, отсутствие симметрий делает дальнейший анализ почти невозможным. Между тем при наличии симметрии задачу, как правило, удастся свести к системе ОДУ.

В действительности, численному решению ДУЧП посвящены целые монографии, к которым мы и отсылаем читателя. В частности, мы рекомендуем превосходную книгу А. А. Самарского. Помимо чисто математического наведения строгости и доказательства теорем (что явно бесполезно с точки зрения физика, работающего на физическом уровне строгости), в тексте можно найти весьма полезные (и зачастую недоказанные) практические рекомендации, опирающиеся на опыт конкретных вычислений.

6.1. Задача Коши для линейных уравнений

Печальный опыт показывает, что в большинстве случаев этот класс задач почему-то начинают решать методами, которые разработаны для обще-

го случая, т. е. для нелинейных уравнений. На самом деле надо вспомнить, что в курсе «Методов математической физики» («Уравнений математической физики») подробнейшим образом излагается, как построить решение в этих задачах. При этом совершенно не важно, о каком именно уравнении идет речь — о гиперболическом волновом уравнении:

$$\partial_t^2 u(x, t) = \partial_x^2 u(x, t) - V(x)u(x, t), \\ u(x, 0) = u_0(x), \quad \partial_t u(x, 0) = w_0(x);$$

о параболическом уравнении теплопроводности:

$$\partial_t u(x, t) = \partial_x^2 u(x, t) - V(x)u(x, t), \quad u(x, 0) = u_0(x)$$

или о нестационарном уравнении Шредингера:

$$i\partial_t \psi(x, t) = -\partial_x^2 \psi(x, t) + V(x)\psi(x, t), \quad \psi(x, 0) = \psi_0(x).$$

Эти задачи могут быть поставлены на конечном интервале с теми или иными граничными условиями, однако обычно они ставятся на всей оси.

Во всех случаях необходимо построить базис собственных функций дифференциального оператора, стоящего в правой части этих уравнений:

$$\lambda_k v_k(x) = -\partial_x^2 v_k(x) + V(x)v_k(x).$$

Разумеется, базис строится с учетом граничных условий; k здесь — непрерывный или дискретный индекс, нумерующий собственные функции. Далее мы будем легкомысленно писать все формулы для случая чисто дискретного спектра, надеясь, что в случае непрерывного спектра читатель сможет заменить сумму на интеграл.

Затем надо разложить начальные условия по построенному базису:

$$u_0(x) = \sum_k c_k v_k(x), \quad w_0(x) = \sum_k d_k v_k(x).$$

После этого пишется явное аналитическое решение задачи — ответ для $u(x, t)$ в произвольный момент времени:

$$u(x, t) = \sum_k c_k(t) v_k(x).$$

При этом для волнового уравнения:

$$c_k(t) = c_k \cos(\sqrt{\lambda_k} t) + d_k \sin(\sqrt{\lambda_k} t) / \sqrt{\lambda_k},$$

для уравнения теплопроводности:

$$c_k(t) = c_k \exp(-\lambda_k t)$$

и для уравнения Шредингера:

$$c_k(t) = c_k \exp(-i\lambda_k t).$$

Обсудим теперь вычислительные аспекты этого рецепта.

Прежде всего, мы (формально говоря) получаем ответ для произвольного момента времени t , притом ошибка в этом ответе практически не зависит от t , она не накапливается с течением времени, что заведомо происходило бы при непосредственном численном счете эволюции системы. Это основное достоинство приведенного рецепта.

Основные трудности при реализации этого рецепта связаны, во-первых, с тем, что спектр дифференциального оператора обычно непрерывный, и, во-вторых, с тем, что задача обычно ставится на бесконечном интервале. Мы в любом случае должны ограничиться конечным числом собственных функций, соответствующих дискретному набору λ_i , начиная от некоторого $\lambda_{\min}$ и заканчивая $\lambda_{\max}$. Мы должны запомнить соответствующие собственные функции $v_i(x)$, для чего необходимо ввести решетку по координате x с шагом dx . Решетка вводится на интервале $[X_-, X_+]$, где, как обычно, $X_{\pm}$ изображают эффективную бесконечность. Заметим, что необязательно брать dx очень маленьким — в промежуточные значения координаты x всегда можно проинтерполировать.

Самое главное, что все эти параметры ($\lambda_{\min}$, $\lambda_{\max}$, число использованных λ_i , $X_{\pm}$ и dx) должны быть согласованы между собой и должны соответствовать поставленной задаче.

Приведем простейший, но очень наглядный пример. Пусть мы решаем задачу рассеяния волнового пакета на потенциальной яме $V(x)$, $V(x \rightarrow \pm\infty) = 0$ для уравнения Шредингера.

Что было бы в отсутствие потенциала?

Мы располагали бы двукратно вырожденным спектром, состоящим из волн $\exp(ikx)$, бегущих вправо, и волн $\exp(-ikx)$, бегущих влево, причем $\lambda_k = -k^2$. Мы собираемся описать движение волнового пакета $\psi(x, t)$, который в начальный момент времени имеет ширину Δx и импульс $k_0 > 0$. Обычно берут стандартный гауссов пакет

$$\psi(x, t = 0) \sim \exp\left(-\frac{(x - x_0)^2}{2\Delta x^2} + ik_0 x\right).$$

Это, действительно, самый разумный выбор, но в принципе можно брать и любой другой пакет.

Поскольку $k_0 > 0$, то пакет бежит вправо. Он представляет собой линейную комбинацию плоских волн

$$\psi(x, t) = \int dk \frac{\exp(ikx)}{\sqrt{2\pi}} \psi(k, t), \quad (*)$$

где волновая функция в импульсном представлении $\psi(k)$ отлична от нуля лишь в окрестности $k = k_0$ шириной порядка Δk , где $\Delta k = 1/\Delta x$. Для гауссова пакета

$$\psi(k, t = 0) \sim \exp\left(-\frac{(k - k_0)^2}{2\Delta k^2} - ikx_0\right),$$

что, действительно, очень удобно с вычислительной точки зрения. Для произвольного пакета

$$\psi(k, t = 0) = \int dx \frac{\exp(-ikx)}{\sqrt{2\pi}} \psi(x, t = 0). \quad (**)$$

Как все это изобразить численно?

На всем диапазоне x , который мы собираемся рассматривать, скажем, $-L < x < L$, следует ввести решетку с шагом dx : $x_n = n dx$. Шаг dx следует выбирать таким, чтобы интеграл $(**)$ более-менее точно изображался интегральной суммой по узлам решетки с шагом dx . В действительности особой точности здесь не требуется, поскольку мелкие детали в форме пакета не очень существенны. Нам гораздо интереснее знать глобальные характеристики пакета: расплывание, запаздывание, искажение формы и появление экспоненциальных хвостов.

Далее мы выбираем диапазон волновых чисел $k \in [k_{\min}, k_{\max}]$, с которым мы будем работать. Для гауссова пакета обычно можно ограничиться диапазоном $k_0 - 3\Delta k < k < k_0 + 3\Delta k$. Для других пакетов диапазон k определяется конкретной формой пакета. На всем диапазоне волновых чисел $[k_{\min}, k_{\max}]$ следует ввести решетку с шагом dk : $k_m = m dk$. Шаг dk , во-первых, должен быть достаточно маленьким для того, чтобы интеграл $(*)$ можно было заменить на интегральную сумму. Во-вторых, он должен быть согласован с диапазоном изменения координаты L . Действительно, сумма

$$\sum_m dk \psi(k_m) \exp(ik_m x) / \sqrt{2\pi} = \sum_m dk \psi(m dk) \exp(i m dk x) / \sqrt{2\pi}$$

есть периодическая функция аргумента x с периодом $P = 2\pi/dk$. Так что если Вы не хотите видеть одновременно два пакета на отрезке $-L < x < L$, период P должен быть меньше $2L$, т. е. $dk < \pi/L$. Если взять $dk = \pi/L$, то после того, как пакет пробежит весь отрезок, часть пакета, исчезающая на правом конце, немедленно начнет высываться на левом конце отрезка.

Итак, по начальным данным, заданным на решетке $\psi(x_n, t = 0)$, вычислим

$$\psi(k_m, t = 0) \simeq \sum_n dx \psi(x_n, t = 0) \exp(-i k_m x_n) / \sqrt{2\pi}.$$

Затем оцениваем, удачно ли мы выбрали dx , dk , $k_{\min}$ и $k_{\max}$. Для этого попросту выполняем «обратную сборку» функции $\psi(x_n, t = 0)$, разложенной по плоским волнам. Величина

$$\sum_m dk \psi(k_m, t = 0) \exp(ik_m x_n) / \sqrt{2\pi}$$

должна не слишком сильно отличаться от исходной $\psi(x_n, t = 0)$. Кроме того, имеет смысл внимательно посмотреть на коэффициенты $\psi(k_m, t = 0)$. Область, в которой они отличны от нуля, должна, с одной стороны, целиком помещаться в отрезок $[k_{\min}, k_{\max}]$. С другой стороны, эта область не должна быть малой долей отрезка $[k_{\min}, k_{\max}]$, иначе мы будем работать с волновыми числами, которые не участвуют в формировании нашего волнового пакета, т. е. будем вычислять суммы, состоящие из огромного количества нулей (точнее, почти нулей).

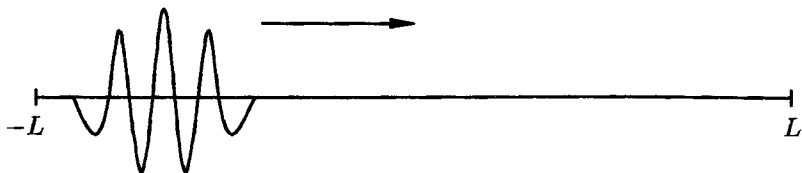
После этого задача нахождения $\psi(x_n, t)$ становится тривиальной:

$$\psi(x_n, t) \simeq \sum_m dk \psi(k_m, t) \exp(ik_m x_n) / \sqrt{2\pi},$$

где

$$\psi(k_m, t) = \psi(k_m, t = 0) \exp(-ik_m^2 t).$$

Так что если Вы возьмете $dk = \pi/L$, выберите начальное условие в виде гауссова пакета:



(здесь нарисована вещественная часть), то он исправно пробежит весь интервал, скроется справа и вынырнет слева. Впрочем, слева он вынырнет уже несколько расплывшимся. С удовольствием напоминаем, что пакет:

- бежит;
- расплывается;
- перебирает своими синусоидальными «лапками» внутри гауссовой огибающей (поскольку фазовая скорость в два раза отличается от групповой).

Студенты-теоретики об этом (обычно) знают, но эту картинку совершенно необходимо хотя бы раз увидеть на экране собственными глазами.

Сделаем маленькое замечание про волновое уравнение.

Если нам нужно решить задачу о движении пакета, который описывается волновым уравнением, то отличие от уравнения Шредингера будет, во-первых, в том, что начальное условие для функции $u(x, t = 0)$ никакого отношения к импульсу пакета не имеет. Так что никаких k_0 в $u(x, t = 0)$ не нужно, достаточно взять

$$u(x, t = 0) \sim \exp\left(-\frac{(x - x_0)^2}{2\Delta x^2}\right).$$

Соответственно, диапазон волновых чисел можно выбрать, скажем, $-3\Delta k < k < +3\Delta k$. Поэтому для волнового уравнения удобнее работать не с экспонентами $\exp(ikx)$, а с вещественными функциями $\sin(kx)$ и $\cos(kx)$.

Импульс пакета определяется начальным условием для $\partial_t u(x, t = 0)$. Чтобы пакет бежал вправо, следует взять $\partial_t u(x, t = 0) = -\partial_x u(x, t = 0)$.

Второе отличие волнового уравнения от уравнения Шредингера будет в том, что зависимость от времени будет другой:

$$u(x_n, t) = \sum_m dk \left[a_m \cos(k_m t) + \frac{b_m}{k_m} \sin(k_m t) \right] \frac{\exp(ik_m x_n)}{\sqrt{2\pi}},$$

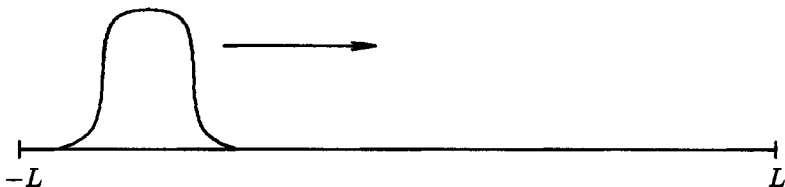
где

$$a_m \simeq \sum_n dx u(x_n, t = 0) \exp(-ik_m x_n) / \sqrt{2\pi},$$

$$b_m \simeq \sum_n dx \partial_0 u(x_n, t = 0) \exp(-ik_m x_n) / \sqrt{2\pi}.$$

Заметим, что для волнового уравнения наличие шага dk приводит не только к периодичности по координате с периодом $2\pi/dk$, но и к периодичности по времени с точно таким же периодом.

Возьмем начальные условия, которые соответствуют гауссовому пакету, расположенному у левого конца интервала и бегущему вправо:



Тогда этот пакет, действительно, побежит вправо, причем его форма никак меняться не будет. За время $2L$ он пробежит весь интервал и начнет исчезать на его правом конце. Однако, ввиду периодичности (как по координате, так и по времени), он тут же начнет высываться на левом конце интервала в совершенно неизменном виде (это как раз и означает периодичность по времени):



Вернемся к уравнению Шредингера. Как описать движение пакета в присутствии потенциала?

Как ни странно, точно таким же образом. Существует единственная чисто техническая трудность. Мы должны разложить волновой пакет не просто по плоским волнам, а по состояниям, описывающим рассеяние на потенциале $V(x)$, т. е. по функциям $\psi_m(x)$ с асимптотиками

$$\psi_m(x \rightarrow -\infty) = \exp(ik_mx) + A_m \exp(-ik_mx),$$

$$\psi_m(x \rightarrow \infty) = B_m \exp(ik_mx). \quad (*)$$

Величина $k_m = mdk$ по-прежнему меняется в диапазоне $k_0 - 3\Delta_k < k_m < k_0 + 3\Delta_k$.

До рассеяния вклад в ответ будет давать только налетающая волна $\exp(ik_m x)$, а после рассеяния — только прошедшая $B_m \exp(ik_m x)$ и отраженная $A_m \exp(-ik_m x)$ волны. Следовательно,

$$\psi(x_n, t) = \sum_m dk c_m \exp(-ik_m^2 t) \psi_m(x_n) / \sqrt{2\pi},$$

где

$$c_m \simeq \sum_n dx \psi(x_n, t=0) \exp(-ik_m x_n) / \sqrt{2\pi}.$$

Так что единственная проблема заключается в том, как численно построить набор функций $\psi_m(x)$.

Каждую из функций $\psi_m(x)$ легко получить, два раза решая задачу Коши (именно задачу Коши, а не краевую задачу) справа налево (именно справа налево) для уравнения

$$\lambda \phi_s(x) = -\partial_x^2 \phi_s(x) + V(x) \phi_s(x)$$

при $\lambda_m = (k_m)^2$, где $s = 1, 2$. Начальные условия в первом случае (при $s = 1$) следует брать:

$$\phi_1(L) = \cos(k_m L), \quad \phi'_1(L) = -k_m \sin(k_m L),$$

а во втором случае (при $s = 2$):

$$\phi_2(L) = \sin(k_m L), \quad \phi'_2(L) = k_m \cos(k_m L).$$

Обе функции следует запоминать в узлах нашей решетки x_n . Кроме того, необходимо запомнить производные на левом конце интервала $\phi'_1(-L)$ и $\phi'_2(-L)$.

Тогда функция $\phi(x) = \phi_1 + i\phi_2$ будет иметь асимптотики

$$\begin{aligned} \phi(x \rightarrow -\infty) &= \frac{1}{B_m} \exp(ik_m x) + \frac{A_m}{B_m} \exp(-ik_m x), \\ \phi(x \rightarrow \infty) &= \exp(ik_m x), \end{aligned}$$

т.е. будет отличаться от искомой $\psi_m(x)$ только множителем $1/B_m$. Этот множитель легко выделить, найдя вронскиан:

$$\begin{aligned} w(\exp(-ik_m x), \phi(x)) &= \frac{2ik_m}{B_m} = \\ &= \exp(-ik_m(-L))\phi'(-L) - (-ik_m) \exp(-ik_m(-L))\phi(-L) \quad (**) \end{aligned}$$

(производные $\phi'_1(-L)$ и $\phi'_2(-L)$ на левом конце интервала мы запоминали как раз ради вычисления вронскиана). Итак, находим из (**) величину B_m , после этого окончательно получаем $\psi_m(x_n) = B_m(\phi_1(x_n) + i\phi_2(x_n))$.

Применение изложенного рецепта приведет к тому, что пакет будет исправно рассеиваться на потенциале — часть пакета будет проходить, а часть — отражаться. Более того, будут прекрасно видны все нетривиальные эффекты: запаздывание; искажение формы; экспоненциальные хвосты, связанные с резонансами (резонансы, они же метастабильные уровни, в англоязычной литературе почему-то называют «квазинормальными модами»).

Сделаем маленькое отступление.

Резонансы есть полюса амплитуд рассеяния, расположенные на комплексной плоскости энергии: $E = E_0 - iE_1$. При этом E_0 определяет энергию, при которой происходит резонанс, а E_1 — ширину резонанса. По совместительству такой полюс есть «почти связанное состояние», т.е. метастабильный уровень, причем E_0 определяет энергию уровня, а E_1 — скорость распада этого уровня. «Волновая функция», соответствующая метастабильному уровню, будет иметь асимптотики

$$\psi(x \rightarrow -\infty) \sim \exp(-i(k_0 - ik_1)x),$$

$$\psi(x \rightarrow \infty) \sim \exp(i(k_0 - ik_1)x),$$

т.е. она экспоненциально растет на бесконечности. Эволюция такого «уровня» имеет вид $\psi(x, t) = \psi(x) \exp(-i(E_0 - iE_1)t)$, а это означает, что он экспоненциально распадается. При этом распад сопровождается излучением бегущей на бесконечность волны, «уносящей вероятность»:

$$\psi(x \rightarrow -\infty, t) \sim \exp(-ik_0x - iE_0t) \exp(-k_1x - E_1t),$$

$$\psi(x \rightarrow \infty, t) \sim \exp(ik_0x - iE_0t) \exp(k_1x - E_1t) \quad (*)$$

(коэффициенты при экспонентах на отрицательной и положительной бесконечностях, вообще говоря, разные). Странноватый на первый взгляд экспоненциальный рост амплитуды этой волны с ростом $|x|$ имеет непосредственный физический смысл: волна, наблюдаемая при большем $|x|$, излучалась раньше. А раз она излучалась раньше, то ее источник (та часть волновой функции, которая живет в окрестности потенциала) имел большую амплитуду (вся волновая функция как целое убывает с течением времени как $\exp(-E_1t)$). Поэтому и сама волна, наблюдаемая при большем $|x|$, должна иметь большую амплитуду.

Хотя резонансы существуют у многих потенциалов (в том числе, как ни странно, даже и у гауссового потенциала $V_0 \exp(-ax^2)$), обычно наиболее ярко выраженные резонансы наблюдаются для потенциалов, которые являются «недовыкопанными» потенциальными ямами. Например, если есть два барьера, то между ними возникает псевдояма, в которой были бы дискретные уровни, если бы барьеры тянулись до бесконечности:



Настоящих дискретных уровней в таком потенциале нет, но есть метастабильные.

Поэтому, во-первых, если поместить частицу с энергией E_0 (разумеется, речь идет о среднем значении энергии) между барьеров, она будет довольно долго там находиться (время жизни $\tau \sim 1/E_1$), медленно просачиваясь сквозь барьеры и убегая на бесконечность в виде плоской волны (*). Подчеркнем, что сама по себе «волновая функция» метастабильного уровня никакого состояния не описывает, она даже не нормируема. Реальное поведение частицы эта функция будет описывать, если ее обрезать (обратить в ноль), начиная с того места, где располагается фронт волны, убегающей на бесконечность. Фронт волны соответствует тому моменту, когда мы посадили частицу между барьерами.

Во-вторых, если рассмотреть рассеяние волнового пакета с энергией E , то при $E = E_0$ будет наблюдаться резонанс (коэффициент прохождения будет иметь максимум шириной порядка E_1 в окрестности энергии $E = E_0$). Формальная причина этого очень проста — близость точки $E = E_0$ к полюсу амплитуды рассеяния $E = E_0 - iE_1$ на комплексной плоскости энергии. Конкретный «динамический» механизм увеличения коэффициента прохождения тоже довольно прост: между барьерами будет «разбужен» метастабильный уровень, и уже после того, как прошедший пакет пройдет, а отраженный отразится, яма еще долго будет «звенеть», излучая на бесконечность медленно затухающую волну, соответствующую энергии E_0 . Тем самым пакеты приобретают экспоненциально затухающие хвосты (шлейфы). Каждый студент-теоретик должен хотя бы раз увидеть на экране эти движущиеся картинки своими глазами.

6.2. Задача Коши для гиперболических уравнений

Еще раз подчеркнем, что мы дадим только минимальные сведения об основных рецептах решения этих задач и об анализе устойчивости разностных схем.

Пусть, мы располагаем системой гиперболических уравнений в частных производных. Раньше всего перепишем ее в терминах первых производных. Довольно ясно, что ее всегда можно свести к системе уравнений вида

$$\partial_t \vec{u}(x, t) = \partial_x \vec{R}(\vec{u}(x, t), x, t) + \vec{A}(\vec{u}(x, t), x, t)$$

(разумеется, такая запись является избыточной: при желании величину $\vec{A}$ можно включить в $\vec{R}$). Рассмотрим для примера известное уравнение Синус-Гордона:

$$\partial_t^2 \phi(x, t) - \partial_x^2 \phi(x, t) + \sin(\phi(x, t)) = 0.$$

Его легко переписать как:

$$\begin{aligned} \partial_t u_1(x, t) &= \partial_x u_2(x, t) + \sin(u_3(x, t)), \\ \partial_t u_2(x, t) &= \partial_x u_1(x, t), \quad \partial_t u_3(x, t) = u_1(x, t), \end{aligned}$$

где

$$\begin{aligned} u_1(x, t) &= \partial_t \phi(x, t), \\ u_2(x, t) &= \partial_x \phi(x, t), \\ u_3(x, t) &= \phi(x, t). \end{aligned}$$

Совершенно аналогично переписываются и более сложные уравнения. Кроме самой системы ДУЧП, нам необходимы еще и начальные условия при $t = 0$ и граничные условия на концах интервала $[x_1, x_2]$.

6.2.1. Самый «наивный» вариант

Если мы введем решетку $x_n = n dx$ по координате x , т.е. перейдем к переменным $\vec{u}_n(t) \equiv \vec{u}(x_n, t)$, то мы получим систему ОДУ, которая может быть решена любым из методов, изложенных в разделе «Задача Коши для ОДУ». Соответствие начальных и граничных условий для исходной системы ДУЧП и полученной системы ОДУ вполне тривиальное.

Это возможный, хотя и не оптимальный вариант.

Необходимо помнить, что динамика решеточной системы не тождественна полевой динамике (см. раздел «Краевая задача для ОДУ»). В решеточной динамике с легкостью появляются эффекты, которые в исходной полевой теории отсутствуют. Кроме того, не следует особо гнаться за

точностью при описании динамики решеточной системы: глупо получать совершенно точный ответ для решеточной динамики, которая не очень точно соответствует нашей исходной модели.

Тем не менее в очень многих работах, посвященных динамике классических нелинейных полей, используют этот довольно наивный подход. Он прост, надежен, но довольно неэффективен — «нарушение симметрии» между временем t и координатой x приводит к лишней возне с переменной t и «превышению точности» по времени. Впрочем, авторы оставляют окончательный выбор за читателем.

6.2.2. Простейший анализ устойчивости

Введем решетку как по времени: $t_m = m dt$, так и по координате: $x_n = n dx$. Введем естественное обозначение $\vec{u}_{n,m} \equiv \vec{u}(n dx, m dt)$. Поскольку обычно не нужно запоминать $\vec{u}(x, t)$ при всех промежуточных временах, то под каждую компоненту $\vec{u}_{n,m}$ отводят одномерный массив. Поэтому мы довольно часто будем опускать временной индекс « m » и значок вектора, т.е. будем писать « u_n » вместо текущего значения $\vec{u}_{n,m}$, писать « $\hat{u}_n$ » вместо нового значения $\vec{u}_{n,m+1}$ и писать « $\check{u}_n$ » вместо старого значения $\vec{u}_{n,m-1}$. При этом под « u_n » понимается либо любая из компонент вектора $\vec{u}_{n,m}$, либо сразу все компоненты этого вектора, если « u_n » стоит в аргументах функций R и A .

Попытаемся изобразить нашу систему ДУЧП

$$\partial_t \vec{u}(x, t) = \partial_x \vec{R}(\vec{u}(x, t), x, t) + \vec{A}(\vec{u}(x, t), x, t)$$

на этой решетке. Первое, что приходит в голову, имеет вид:

$$\hat{u}_n = u_n + dt \left[\frac{R(u_{n+1}, x_{n+1}, t_m)}{2dx} - \frac{R(u_{n-1}, x_{n-1}, t_m)}{2dx} + A(u_n, x_n, t_m) \right]. \quad (*)$$

В этой схеме есть много несуразностей, самая простая и очевидная состоит в том, что у нее разный порядок по времени (первый) и по координате (второй).

Главный же недостаток этой схемы заключается в том, что никакого отношения к истинной эволюции системы соотношение (*) не имеет. Действительно, исследуем устойчивость этой схемы относительно малых возмущений. Рассмотрим простейший случай, когда вектор $\vec{u}_{n,m}$ состоит

всего из одной компоненты. Дадим величинам u_n малые приращения δu_n и линеаризуем (*) относительно δu_n :

$$\delta \hat{u}_n = \delta u_n + dt \left[\frac{\partial_{u_{n+1}} R(u_{n+1}, x_{n+1}, t_m)}{2dx} \delta u_{n+1} - \frac{\partial_{u_{n-1}} R(u_{n-1}, x_{n-1}, t_m)}{2dx} \delta u_{n-1} + \partial_{u_n} A(u_n, x_n, t_m) \delta u_n \right] \quad (**)$$

Рассмотрим эволюцию δu_n в пределах небольшого участка решетки, на котором $\partial_u R(u, x, t)$ можно полагать почти постоянным и равным $\mathcal{R}'$. Аналогично, полагаем $\partial_u A(u, x, t)$ почти постоянным и равным $\mathcal{A}'$. Ввиду линейности линеаризованного уравнения (**), величину δu_n следует разложить по плоским волнам и рассматривать эволюцию каждой волны отдельно. Выясним, как связана амплитуда a плоской волны $\delta u_n = a \exp(ikx_n)$ в текущий момент времени с амплитудой $\hat{a}$ этой же волны $\delta \hat{u}_n = \hat{a} \exp(ikx_n)$ на следующем шаге по времени. Из уравнения (**) получаем:

$$\hat{a} = a + \frac{dt}{dx} \left[a \mathcal{R}' \frac{\exp(ikdx) - \exp(-ikdx)}{2} + a \mathcal{A}' dx \right].$$

Обычно величины $\mathcal{R}'$ и $\mathcal{A}'$ не слишком отличаются по абсолютной величине. Тогда слагаемым $\mathcal{A}'$ можно пренебречь, ведь оно умножается на dx . Окончательно получаем:

$$\hat{a} = a[1 + i\alpha \sin(k dx)],$$

где введено традиционное обозначение $\alpha \equiv \mathcal{R}' dt/dx$. Полученный ответ для $\hat{a}$ означает крушение придуманной нами схемы, поскольку все гармоники малого возмущения на каждом шаге растут по модулю:

$$|\hat{a}/a| = \sqrt{1 + \alpha^2 \sin^2(k dx)} > 1,$$

иными словами имеет место экспоненциальный рост возмущения со временем. «Веселее» всего растут гармоники с $k = \pm\pi/(2dx)$, т. е. имеющие период в 4 узла решетки. Довольно ясно, что при любых начальных условиях мы очень скоро получим некую комбинацию гармоник с $k \approx \pm\pi/(2dx)$ со всевозрастающей амплитудой, а затем и переполнение.

Чтобы продемонстрировать, от какой малости зависит устойчивость или неустойчивость схемы, приведем пример такой же неудачной, зато

устойчивой схемы:

$$\hat{u}_n = \frac{1}{2} [u_{n-1} + u_{n+1}] + dt \left[\frac{R(u_{n+1}, x_{n+1}, t_m) - R(u_{n-1}, x_{n-1}, t_m)}{2dx} + A(u_n, x_n, t_m) \right].$$

Для этой схемы получится:

$$|\hat{a}/a|^2 = \cos^2(k dx) + \alpha^2 \sin^2(k dx) = 1 + (\alpha^2 - 1) \sin^2(k dx),$$

т.е. схема устойчива при $\alpha < 1$, что означает $dt < dx/\mathcal{R}'$. Поскольку для волновых уравнений величина $\mathcal{R}'$ — это что-то вроде скорости распространения сигнала, то обычно произносят слова о том, что «сигнал должен успеть добежать от соседнего узла до текущего узла, поэтому dt ограничено». Вряд ли это хорошее объяснение. С равным успехом условие $dt < dx/\mathcal{R}'$ можно получить просто по размерности.

6.2.3. Простейший удовлетворительный рецепт

Итак, нам необходимо написать устойчивую схему, которая имеет второй порядок как по времени, так и по координате. Одной из таких схем является полушаговая схема, которую мы и рекомендуем для гиперболических задач. Она в некотором роде аналогична схеме Рунге-Кутты второго порядка и требует одного дополнительного массива для записи полушаговых данных.

Прежде всего, делается полшага по времени (т.е. на $dt/2$), причем вычисляются значения функции $u(x, t + dt/2)$ не в узлах решетки, а в точности посередине между узлами:

$$u_{n+1/2} = \frac{1}{2} [u_n + u_{n+1}] + (dt/2) \left[\frac{R(u_{n+1}, x_{n+1}, t_m) - R(u_n, x_n, t_m)}{dx} + \frac{1}{2} [A(u_n, x_n, t_m) + A(u_{n+1}, x_{n+1}, t_m)] \right].$$

А уже затем вычисляются значения $\hat{u}_n$:

$$\hat{u}_n = u_n + (dt) \left[\frac{R(u_{n+1/2}, x_{n+1/2}, t_m) - R(u_{n-1/2}, x_{n-1/2}, t_m)}{dx} + \frac{1}{2} [A(u_{n-1/2}, x_{n-1/2}, t_m) + A(u_{n+1/2}, x_{n+1/2}, t_m)] \right].$$

Анализ устойчивости дает следующее выражение для $\hat{a}$:

$$\hat{a} = a \left[1 + \alpha \left(i \sin(k dx) + \alpha (\cos(k dx) - 1) \right) \right],$$

откуда

$$\begin{aligned} |\hat{a}/a|^2 &= 1 + \alpha^4 (\cos(k dx) - 1)^2 - \alpha^2 (\cos(k dx) - 1)^2 = \\ &= 1 + \alpha^2 (\alpha^2 - 1) (1 - \cos(k dx))^2. \end{aligned}$$

Иными словами, схема устойчива при условии $\alpha < 1$.

Еще раз напомним, что в использованной нами несколько условной записи опущены индексы, нумерующие компоненты векторов $\vec{u}_{n,m}$, $\vec{R}$ и $\vec{A}$.

Иногда в литературе можно встретить двухслойную схему:

$$\hat{u}_n = \check{u}_n + (2dt) \left[\frac{R(u_{n+1}, x_{n+1}, t_m) - R(u_{n-1}, x_{n-1}, t_m)}{2dx} + A(u_n, x_n, t_m) \right].$$

Это тоже схема второго порядка, она устойчива при условии $\alpha < 1$, но мы не рекомендуем ее применять. Дело в том, что двухслойная схема имеет очень своеобразный дефект: вся двумерная решетка (x_n, t_m) расслаивается на две подрешетки: узлы первой удовлетворяют условию $(-)^{n+m} = 1$, узлы второй — условию $(-)^{n+m} = -1$. Обе подрешетки в двухслойной схеме эволюционируют совершенно независимо друг от друга и, как показывает печальный опыт, «разъезжаются» в разные стороны.

6.2.4. Больше число измерений

До сих пор мы работали с гиперболическими уравнениями в двумерном (1+1) пространстве. Обобщение приведенных выше рецептов на случай (2+1), (3+1), ... (n+1) сделать не так уж сложно. Собственно, достаточно разобраться со случаем (2+1), а более высокие размерности обрабатываются совершенно аналогично. Прежде всего перепишем нашу систему ДУЧП так, чтобы в ней остались только первые производные:

$$\begin{aligned} \partial \vec{u}(x, y, t) &= \partial_x \vec{R}_x(\vec{u}(x, y, t), x, y, t) + \partial_y \vec{R}_y(\vec{u}(x, y, t), x, y, t) + \\ &+ \vec{A}(\vec{u}(x, y, t), x, y, t). \end{aligned}$$

Введем решетку (x_n, y_m, t_k) , причем, просто чтобы не вводить лишние буквы, будем считать, что шаг решетки по x совпадает с шагом решетки по y : $dx = dy$. Опять-таки, просто чтобы укоротить формулы, выкинем

явную зависимость от x , y и t у функций R_x , R_y и A . Кроме того, как и в двумерном случае, опустим временной индекс и значок вектора: вместо $\vec{u}_{n,m,k}$ будем писать $u_{n,m}$, а вместо $\vec{u}_{n,m,k+1}$ будем писать $\hat{u}_{n,m}$.

Очевидно, что анализ устойчивости проводится точно так же, как и в двумерном пространстве. Правда, у плоских волн, по которым раскладывается возмущение, волновое число будет вектором с компонентами k_x и k_y :

$$\delta u_{n,m} = a \exp(ik_x x_n + ik_y y_m) = a \exp(ik_x n dx + ik_y m dx).$$

Кстати, обычно можно полагать $k_x = k_y$ — если гармоника плоха по оси x , она, скорее всего, будет так же плоха и по оси y .

Начнем с неудачной, зато устойчивой схемы:

$$\hat{u}_{n,m} = \frac{1}{4} [u_{n-1,m} + u_{n+1,m} + u_{n,m+1} + u_{n,m-1}] + \\ + dt \left[\frac{R_x(u_{n+1,m}) - R_x(u_{n-1,m})}{2dx} + \frac{R_x(u_{n,m+1}) - R_x(u_{n,m-1})}{2dx} + A(u_{n,m}) \right].$$

Как и в двумерном (1+1) случае, устойчивость достигается «за счет соседей». Совершенно ясно, что если вместо среднего арифметического соседних узлов

$$\frac{1}{4} [u_{n-1,m} + u_{n+1,m} + u_{n,m+1} + u_{n,m-1}]$$

написать просто $u_{n,m}$, то амплитуда возмущения с любым волновым числом начнет экспоненциально расти:

$$\hat{a}/a = 1 + \alpha i \sin(k_x dx) + \alpha i \sin(k_y dx).$$

А вот если мы «цепляемся за соседей», то получается:

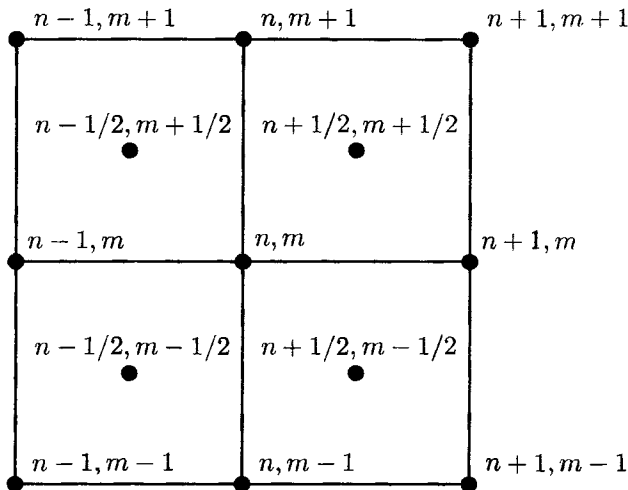
$$\hat{a}/a = \cos(k_x dx)/2 + \cos(k_y dx)/2 + \alpha i \sin(k_x dx) + \alpha i \sin(k_y dx) = \\ = \cos((k_x - k_y)dx/2) \left[\cos((k_x + k_y)dx/2) + 2i\alpha \sin((k_x + k_y)dx/2) \right],$$

откуда

$$|\hat{a}/a|^2 = \cos^2((k_x - k_y)dx/2) \left[1 + (4\alpha^2 - 1) \sin^2((k_x + k_y)dx/2) \right].$$

Как уже было сказано, хуже всего этой схеме приходится при одинаковых k_x и k_y : $k_x = k_y = \pi/(2dx)$. Тем не менее, при $\alpha < 1/2$ схема устойчива.

Соорудим теперь аналог полшаговой схемы, который мы и рекомендуем к использованию. Прежде всего, сдвигаемся на полшага по времени ($dt/2$) и вычисляем значения u в серединках квадратиков, примыкающих к узлу (x_n, y_m) .



Эти четыре значения вычисляются так:

$$\begin{aligned}
 u_{n+1/2, m+1/2} = & \frac{1}{4} [u_{n, m} + u_{n+1, m} + u_{n, m+1} + u_{n+1, m+1}] + \\
 & + (dt/2) \left[\frac{1}{2} \left(\frac{R_x(u_{n+1, m}) - R_x(u_{n, m})}{dx} + \frac{R_x(u_{n+1, m+1}) - R_x(u_{n, m+1})}{dx} \right) + \right. \\
 & + \frac{1}{2} \left(\frac{R_y(u_{n, m+1}) - R_y(u_{n, m})}{dx} + \frac{R_y(u_{n+1, m+1}) - R_y(u_{n+1, m})}{dx} \right) + \\
 & \left. + \frac{1}{4} [A(u_{n, m}) + A(u_{n+1, m}) + A(u_{n, m+1}) + A(u_{n+1, m+1})] \right].
 \end{aligned}$$

Остальные три соотношения легко получить, заменяя пару индексов (n, m) на $(n-1, m)$, $(n, m-1)$ и $(n-1, m-1)$. После этого делаем шаг на dt для

текущего узла (n, m) :

$$\begin{aligned} \hat{u}_{n,m} = & \frac{1}{4} \left[u_{n+1/2,m+1/2} + u_{n-1/2,m+1/2} + u_{n+1/2,m-1/2} + u_{n-1/2,m-1/2} \right] + \\ & + (dt/2) \left[\frac{1}{2} \left(\frac{R_x(u_{n+1/2,m+1/2}) - R_x(u_{n-1/2,m+1/2})}{dx} + \right. \right. \\ & \left. \left. + \frac{R_x(u_{n+1/2,m-1/2}) - R_x(u_{n-1/2,m-1/2})}{dx} \right) + \right. \\ & \left. + \frac{1}{2} \left(\frac{R_y(u_{n+1/2,m+1/2}) - R_y(u_{n+1/2,m-1/2})}{dy} + \right. \right. \\ & \left. \left. + \frac{R_y(u_{n-1/2,m+1/2}) - R_y(u_{n-1/2,m-1/2})}{dy} \right) + \right. \\ & \left. + \frac{1}{4} \left[A(u_{n+1/2,m+1/2}) + A(u_{n-1/2,m+1/2}) + A(u_{n+1/2,m-1/2}) + \right. \right. \\ & \left. \left. + A(u_{n-1/2,m-1/2}) \right] \right]. \end{aligned}$$

Эта схема представляет собой двукратно примененную одномерную полушаговую схему, поэтому анализ устойчивости ничего нового не даст: $\alpha < 1/2$. Распространить эту схему на случай $(3+1)$ никакого труда не представляет. Может возникнуть вопрос, отчего мы не брали в качестве «полушаговых точек» середины ребер нашей сетки, т. е. точки $(n, m \pm 1/2)$ и $(n \pm 1/2, m)$. В общем-то, это вполне возможный вариант, но тогда пришлось бы вычислять производные по x и y на интервале $2dx$, скажем, для точки $(n, m + 1/2)$ пришлось бы писать:

$$\frac{1}{2} \frac{R_x(u_{n+1,m+1}) - R_x(u_{n-1,m+1})}{2dx} + \frac{1}{2} \frac{R_x(u_{n+1,m}) - R_x(u_{n-1,m})}{2dx},$$

так что этой идеей лучше не пользоваться.

6.3. Задача Коши для параболических уравнений

Как ни странно, нам придется различать параболические уравнения типа Шредингера

$$i\partial_t\psi(x,t) = -\partial_x^2\psi(x,t) + V(\psi(x,t))$$

и параболические уравнения теплопроводности

$$\partial_t u(x, t) = \partial_x [D(u(x, t)) \partial_x u(x, t)].$$

Для этих двух классов уравнений приходится рекомендовать слегка различающиеся методы.

Перепишем параболическое уравнение в общем виде. Разумеется, в отличие от гиперболических уравнений, от второй производной по координате x тут не избавишься:

$$\partial_t u(x, t) = \partial_x R(\partial_x u(x, t), u(x, t), x, t) + A(u(x, t), x, t).$$

Как и для гиперболических уравнений, эта запись весьма избыточна, от величины A здесь можно избавиться. Заметим, что неизвестная функция $u(x, t)$ здесь однокомпонентна. Про случай нескольких компонент $\vec{u}(x, t)$ мы сделаем отдельное замечание.

6.3.1. Анализ устойчивости

Как обычно, начнем с простейшей схемы:

$$\hat{u}_n = u_n + dt \left[\frac{1}{dx} \left(R\left(\frac{u_{n+1} - u_n}{dx}, \frac{u_{n+1} + u_n}{2}, \frac{x_{n+1} + x_n}{2}, t_m\right) - R\left(\frac{u_{n+1} - u_n}{dx}, \frac{u_{n+1} + u_n}{2}, \frac{x_{n+1} + x_n}{2}, t_m\right) \right) + A(u_n, x_n, t_m) \right].$$

При анализе устойчивости (подробности см. в разделе «Задача Коши для гиперболических уравнений») основной вклад даст слагаемое с dx^2 в знаменателе (оно соответствует второй производной). Как и ранее, введем обозначение $\mathcal{R}'$ для производной R по первому аргументу. Тогда амплитуда возмущения с волновым числом k на следующем шаге будет равна:

$$\hat{a} = a + a \frac{dt}{dx^2} \mathcal{R}' (2 \cos(k dx) - 2).$$

Мы будем полагать, что $\mathcal{R}' > 0$,¹⁰ поэтому $\hat{a}/a$ всегда меньше единицы. Однако, может оказаться, что $\hat{a}/a < -1$, а это тоже дает экспоненциальный

¹⁰Случай $\mathcal{R}' < 0$ соответствует «взрывающемуся» уравнению, в котором все гармоники действительно экспоненциально растут. Этот рост есть свойство уравнения, а не разностной схемы. Мы же рассматриваем случай, когда уравнение не «взрывается», а разностная схема неустойчива («взрывается»).

рост для амплитуды малого возмущения. При этом «веселее» всего будет расти гармоника с $k = \pi/dx$, что соответствует поведению $\delta u_n = (-1)^n$. Следовательно, необходимо потребовать $dt\mathcal{R}'/dx^2 < 1/2$. Это означает катастрофу. Шаг по координате dx должен быть маленьким, чтобы получить удовлетворительную точность для производной ∂_x . Но тогда шаг по времени должен быть совсем микроскопический — порядка dx^2 . Заметим, что это ограничение накладывается из-за таких решеточных степеней свободы, которые к исходной непрерывной системе вообще никакого отношения не имеют, а именно из-за волн вида $\delta u_n = (-1)^n$.

6.3.2. Два простейших рецепта

За спасение от катастрофы, описанной в предыдущем разделе, придется заплатить дорогую цену: мы будем использовать неявную схему. Это значит, что мы получим не явное выражение для $\hat{u}_n$, а нелинейное уравнение для $\hat{u}_n$, которое еще придется решать.

Первый вариант, который мы рекомендуем для уравнений типа Шредингера:

$$\begin{aligned} \hat{u}_n = u_n + \frac{1}{2} dt \left[\frac{1}{dx} \left(R\left(\frac{\hat{u}_{n+1} - \hat{u}_n}{dx}, \frac{\hat{u}_{n+1} + \hat{u}_n}{2}, \frac{x_{n+1} + x_n}{2}, t_{m+1}\right) - \right. \right. \\ \left. \left. - R\left(\frac{\hat{u}_{n+1} - \hat{u}_n}{dx}, \frac{\hat{u}_{n+1} + \hat{u}_n}{2}, \frac{x_{n+1} + x_n}{2}, t_{m+1}\right) \right) + A(\hat{u}_n, x_n, t_{m+1}) \right] + \\ + \frac{1}{2} dt \left[\frac{1}{dx} \left(R\left(\frac{u_{n+1} - u_n}{dx}, \frac{u_{n+1} + u_n}{2}, \frac{x_{n+1} + x_n}{2}, t_m\right) - \right. \right. \\ \left. \left. - R\left(\frac{u_{n+1} - u_n}{dx}, \frac{u_{n+1} + u_n}{2}, \frac{x_{n+1} + x_n}{2}, t_m\right) \right) + A(u_n, x_n, t_m) \right]. \end{aligned}$$

Эта схема замечательна тем, что она второго порядка точности по времени. Проверим ее устойчивость:

$$\hat{a} = a + \hat{a} \frac{dt}{2dx^2} \mathcal{R}'(2 \cos(kdx) - 2) + a \frac{dt}{2dx^2} \mathcal{R}'(2 \cos(kdx) - 2),$$

откуда

$$\hat{a}/a = \frac{1-y}{1+y}, \quad \text{где} \quad y = \frac{dt}{dx^2} \mathcal{R}'(1 - \cos(kdx)).$$

Поскольку $y > 0$, то $|\hat{a}/a| < 1$ без каких-либо условий на dt . Разумеется, слишком большой шаг по времени dt взять невозможно — мы получим

недостаточно точный ответ при взятии разностной производной по времени, но соображениями устойчивости шаг dt не ограничен.

Казалось бы, этим вариантом и надо всегда пользоваться. Ничего подобного.

Второй вариант, который мы рекомендуем для уравнений переноса:

$$\hat{u}_n = u_n + dt \left[\frac{1}{dx} \left(R \left(\frac{\hat{u}_{n+1} - \hat{u}_n}{dx}, \frac{\hat{u}_{n+1} + \hat{u}_n}{2}, \frac{x_{n+1} + x_n}{2}, t_{m+1} \right) - \right. \right. \\ \left. \left. - R \left(\frac{\hat{u}_{n+1} - \hat{u}_n}{dx}, \frac{\hat{u}_{n+1} + \hat{u}_n}{2}, \frac{x_{n+1} + x_n}{2}, t_{m+1} \right) \right) + A(\hat{u}_n, x_n, t_{m+1}) \right].$$

Хотя эта схема на один порядок менее точная, чем предыдущая, как показывает опыт, для уравнений переноса надо использовать именно ее. Обратите внимание, что даже там, где это не влияет на устойчивость, а именно во втором аргументе R , стоит $\hat{u}$. Опять-таки, это рецепт, проверенный экспериментально.

Теперь обсудим, как решать полученную систему нелинейных уравнений относительно $\hat{u}_n$. И в первом, и во втором варианте используется один и тот же метод — метод Ньютона (см. часть I, раздел «Многомерные корни»). К счастью, та система линейных уравнений, которую нам здесь придется решать, будет тридиагональной — ведь в n -ом уравнении присутствуют только $\hat{u}_{n-1}$, $\hat{u}_n$ и $\hat{u}_{n+1}$. Иными словами, n -ый узел решетки «запутан» только со своими ближайшими соседями — с $(n-1)$ -ым и $(n+1)$ -ым узлами. Это означает, что количество операций, потраченных на одну итерацию в методе Ньютона, пропорционально первой степени N (N — количество узлов решетки по переменной x). В качестве начального приближения берется u_n . Опыт показывает, что метод Ньютона сходится к $\hat{u}_n$ за 5–6 итераций.

Итак, чтобы один раз шагнуть по времени на dt для параболического уравнения, надо 5–6 раз решить тридиагональную СЛУ. Это требует заметно большего числа операций, чем один шаг по времени для гиперболических уравнений.

Сделаем существенное замечание.

При применении неявных схем приходится решать систему нелинейных уравнений. Ее вид таков, что применение итерационного метода прямо-таки напрашивается. Действительно, величины $\hat{u}_n$ заведомо приблизительно равны u_n , поскольку мы делаем один шаг по времени на маленькую величину dt . Следовательно, мы располагаем хорошим начальным приближением к корню. Кроме того, система написана как раз в подходящем виде

(система в сущности выглядит как $\hat{u}_n = F(\hat{u}_n)$). Поэтому, казалось бы, полагаем $\hat{u}_n^{(0)} = u_n$, а затем последовательно вычисляем $\hat{u}_n^{(k+1)} = F(\hat{u}_n^{(k)})$, пока $\hat{u}_n^{(k+1)}$ не станет совпадать с $\hat{u}_n^{(k)}$.

Увы, опыт показывает, что если для уравнения необходимо применять неявную схему, то метод итераций никогда не сходится. Так что для неявных схем можно использовать только метод Ньютона, а метод итераций применять нельзя.

Наконец, сделаем замечание про случай нескольких компонент $\vec{u}(x, t)$. Собственно, никаких принципиальных отличий от однокомпонентного случая нет, но зато есть довольно существенное техническое усложнение: матрица СЛУ, которая появится в методе Ньютона, будет не тридиагональной, а блочно-диагональной. Способ решения такой СЛУ довольно подробно изложен в разделе «Краевая задача ОДУ. Релаксационные методы».

6.3.3. Больше число измерений

К сожалению, здесь нас ждет не очень приятный сюрприз. Довольно ясно, что анализ устойчивости для размерности (2+1) даст приблизительно то же самое, что получилось в случае (1+1). Следовательно, неявной схемы не избежать. Кстати, чтобы избежать уж очень громоздких записей, мы сильно сократим наше параболическое уравнение:

$$\left. \begin{aligned} i\partial_t u(x, y, t) \\ \partial_t u(x, y, t) \end{aligned} \right\} = \partial_x R_x(\partial_x u(x, y, t)) + \partial_y R_y(\partial_y u(x, y, t)).$$

Кроме того, мы ограничимся рецептом для уравнения теплопроводности. Этот рецепт легко распространить как на случай уравнения Шредингера, так и на случай более сложного уравнения.¹¹

Теперь о неприятном сюрпризе. Если мы просто напишем:

$$\begin{aligned} \hat{u}_{n,m} = u_{n,m} + dt \left[\frac{1}{dx} \left(R_x \left(\frac{\hat{u}_{n+1,m} - \hat{u}_{n,m}}{dx} \right) - R_x \left(\frac{\hat{u}_{n+1,m} - \hat{u}_{n,m}}{dx} \right) \right) + \right. \\ \left. + \frac{1}{dx} \left(R_y \left(\frac{\hat{u}_{n,m+1} - \hat{u}_{n,m}}{dx} \right) - R_y \left(\frac{\hat{u}_{n,m+1} - \hat{u}_{n,m}}{dx} \right) \right) \right], \end{aligned}$$

то мы не сможем применить метод Ньютона. Дело в том, что в этой системе уравнений узел (n, m) запутан с четырьмя соседними узлами — $(n-1, m)$, $(n+1, m)$, $(n, m-1)$ и $(n, m+1)$. Поэтому упорядочить узлы так, чтобы

¹¹Рецепт можно применять, если в уравнении можно расцепить вторые производные по осям x и y .

матрица СЛУ, которая появляется в методе Ньютона, стала тридиагональной (или хотя бы блочно-диагональной, см. раздел «Краевая задача ОДУ. Релаксационные методы») невозможно. Поэтому если мы применим метод Ньютона, то на каждой итерации нам придется решать СЛУ общего вида, а это совершенно нереально, поскольку размер матрицы СЛУ будет $N^2 \times N^2$ (мы полагаем, что количество узлов в решетках по осям x и y одинаково: $N_x = N_y = N$).

Выход из положения совершенно фантастический.

Сделаем один шаг по времени так, как будто второй производной по y нет вовсе:

$$\tilde{u}_{n,m} = u_{n,m} + dt \left[\frac{1}{dx} \left(R_x \left(\frac{\tilde{u}_{n+1,m} - \tilde{u}_{n,m}}{dx} \right) - R_x \left(\frac{\tilde{u}_{n+1,m} - \tilde{u}_{n,m}}{dx} \right) \right) \right].$$

Затем сделаем второй шаг по времени так, как будто нет второй производной по x :

$$\hat{u}_{n,m} = \tilde{u}_{n,m} + dt \left[\frac{1}{dx} \left(R_y \left(\frac{\hat{u}_{n,m+1} - \hat{u}_{n,m}}{dx} \right) - R_y \left(\frac{\hat{u}_{n,m+1} - \hat{u}_{n,m}}{dx} \right) \right) \right].$$

На первом шаге вся задача делится на слои, параллельные оси x . Для каждого фиксированного y_m мы получаем одномерное параболическое уравнение, решение которого подробно описано в предыдущем разделе. Точно так же, на втором шаге вся задача делится на слои, параллельные оси y , причем величины $u_{n_1,m}$ и $u_{n_2,m}$ из двух разных слоев с разными x_{n_1} и x_{n_2} эволюционируют совершенно независимо.

Трудно поверить, что такой рецепт может давать верный ответ, однако опыт показывает, что он вполне применим.

Более того, и в совсем общем случае, когда в правой части системы ДУЧП стоит сумма некоторых конструкций $A_1, \dots, A_n$, эту правую часть можно обрабатывать пошагово. Именно, один шаг по времени на dt разбивается на n шагов. Сначала надо сделать первый шаг с помощью A_1 (игнорируя остальные A_i), затем — второй шаг с помощью A_2 и т. д., вплоть до n -го шага, который делается с помощью A_n . Более того, для разных шагов можно применять разные схемы, т. е., делая i -ый шаг, мы можем выбрать ту схему, которая лучше подходит для данного A_i .

6.4. Краевая задача для линейных уравнений

Очевидно, что краевая задача для линейных уравнений с делящимися переменными должна решаться точно так же, как и задача Коши для линей-

ных уравнений, т. е. мы должны численно реализовать тот метод, который предписан теорией ДУЧП. В качестве примера рассмотрим уравнение Гельмгольца с правой частью (источником) в цилиндре:¹²

$$\Delta u(r, \varphi, z) + k^2 u(r, \varphi, z) = \rho(r, \varphi, z), \quad r \in [0, a], \quad \varphi \in [0, 2\pi], \quad z \in [0, H].$$

Пусть граничные условия имеют вид:

$$u(r = a, \varphi, z) = f_1(\varphi, z),$$

$$u(r, \varphi, z = 0) = f_2(r, \varphi),$$

$$u(r, \varphi, z = H) = f_3(r, \varphi).$$

Решение этой задачи следует искать в виде:

$$u(r, \varphi, z) = u^{(1)}(r, \varphi, z) + u^{(2)}(r, \varphi, z) + u^{(3)}(r, \varphi, z) + u^{(4)}(r, \varphi, z).$$

При этом

$$u^{(1)}(r, \varphi, z) = \sum_{l=1}^{\infty} \sum_{m=-\infty}^{\infty} \sum_{n=1}^{\infty} C_{lmn}^{(1)} \frac{e^{im\varphi}}{2\pi} N_{mn} J_{|m|}(k_{mn}r) \sqrt{\frac{2}{H}} \sin(p_l z),$$

$$u^{(2)}(r, \varphi, z) = \sum_{l=1}^{\infty} \sum_{m=-\infty}^{\infty} C_{lm}^{(2)} \frac{e^{im\varphi}}{2\pi} J_{|m|}(K_l r) \sqrt{\frac{2}{H}} \sin(p_l z),$$

$$u^{(3)}(r, \varphi, z) = \sum_{m=-\infty}^{\infty} \sum_{n=1}^{\infty} C_{mn}^{(3)} \frac{e^{im\varphi}}{2\pi} N_{mn} J_{|m|}(k_{mn}r) \operatorname{sh}(P_{mn}z),$$

$$u^{(4)}(r, \varphi, z) = \sum_{m=-\infty}^{\infty} \sum_{n=1}^{\infty} C_{mn}^{(4)} \frac{e^{im\varphi}}{2\pi} N_{mn} J_{|m|}(k_{mn}r) \operatorname{sh}(P_{mn}(H-z)).$$

Здесь:

- $p_l = \pi l / H$;
- $k_{mn} = \mu_{mn} / a$, где μ_{mn} — n -ый корень функции Бесселя $J_{|m|}(x)$; N_{mn} — нормировочный множитель для функции Бесселя на отрезке $[0, a]$, т. е.

$$N_{mn}^2 \int_0^a r \, dr \, J_m^2(k_{mn}r) = N_{mn}^2 a^2 J'_m(k_{mn}a) / 2 = 1;$$

¹²Мы намеренно выбрали довольно нетипичный пример.

- $K_l = \sqrt{k^2 - p_l^2}$ (для мнимых K_l вместо функции Бесселя $J_{|m|}(x)$ следует брать функцию Инфельда $I_{|m|}(x)$);
- $P_{mn} = \sqrt{k_{mn}^2 - k^2}$ (для мнимых P_{mn} вместо $\text{sh}(x)$ следует брать $\sin(x)$).

Коэффициенты $C^{(a)}$ определяются из соотношений:

$$C_{lmn}^{(1)}(-k_{mn}^2 - p_l^2 + k^2) = \rho_{lmn},$$

$$C_{lm}^{(2)} J_m(K_l a) = f_{lm}^{(1)},$$

$$C_{mn}^{(3)} \text{sh}(P_{mn} H) = f_{mn}^{(3)},$$

$$C_{mn}^{(4)} \text{sh}(P_{mn} H) = f_{mn}^{(2)},$$

здесь

$$\rho_{lmn} = \int_0^a r dr \int_0^{2\pi} d\varphi \int_0^H dz \frac{e^{im\varphi}}{2\pi} N_{mn} J_{|m|}(k_{mn} r) \sqrt{\frac{2}{H}} \sin(p_l z) \rho(r, \varphi, z),$$

$$f_{lm}^{(1)} = \int_0^{2\pi} d\varphi \int_0^H dz \frac{e^{im\varphi}}{2\pi} \sqrt{\frac{2}{H}} \sin(p_l z) f^{(1)}(\varphi, z),$$

$$f_{mn}^{(2)} = \int_0^a r dr \int_0^{2\pi} d\varphi \frac{e^{im\varphi}}{2\pi} N_{mn} J_{|m|}(k_{mn} r) f^{(2)}(r, \varphi),$$

$$f_{mn}^{(3)} = \int_0^a r dr \int_0^{2\pi} d\varphi \frac{e^{im\varphi}}{2\pi} N_{mn} J_{|m|}(k_{mn} r) f^{(3)}(r, \varphi).$$

Таков довольно громоздкий, но вполне тривиальный рецепт аналитического решения этой задачи.

Как ни странно, реализовать этот рецепт численно тоже довольно просто. Раньше всего следует ограничить число используемых базисных функций — ясно, что для любой достаточно гладкой функции¹³ коэффициенты достаточно быстро убывают с ростом любого из индексов, так что можно ограничиться конечным числом слагаемых. После этого аналитически (либо

¹³Неаналитичности в функциях лучше обрабатывать аналитически. Можно выделить разрыв в функции, пропорциональный $\theta(x)$, или разрыв в производной, пропорциональный $|x|$, в явной аналитической форме. После этого «довесок» к этой неаналитичности, т. е. достаточно гладкую функцию, можно искать численно.

непосредственным численным интегрированием) вычисляем коэффициенты $C^{(a)}$, затем суммируем ряды (точнее, отрезки рядов) для функций $u^{(a)}$ и получаем окончательный ответ $u(r, \varphi, z)$.

6.5. Краевая задача для эллиптических уравнений

Как можно было бы решать краевую задачу для нелинейного эллиптического уравнения?

К сожалению, опыт решения краевой задачи для ОДУ нам не пригодится. Во-первых, метод «стрельбы» можно реализовать только в одномерном случае. Во-вторых, применить тот метод, который использовался в релаксационных методах для ОДУ, т. е. поиск многомерного корня методом Ньютона, здесь тоже не получится. Дело в том, что на каждом обороте метода Ньютона надо решать СЛУ. В одномерном случае матрица этой СЛУ была блочно-диагональной, что и позволяло решить ее за конечное время. При большем числе измерений матрица не будет блочно-диагональной, в результате мы столкнемся с явно нереальной задачей — решать СЛУ общего вида с матрицей размера $N^2 \times N^2$ (N — количество узлов в решетках по осям x и y).

Поэтому, в сущности, метод решения краевой задачи для ДУЧП существует только один: мы пишем некоторую динамическую задачу, результатом эволюции которой является искомое решение. Классические методы соответствуют написанию параболического уравнения. Методы минимизации невязки соответствуют написанию гиперболического уравнения с трением (минимум невязки или другого функционала ищется динамическим методом).

Чтобы избежать очень длинных формул, мы будем писать все рецепты на примере простейшего двумерного нелинейного уравнения

$$\Delta u(x, y) + V(u(x, y)) = 0.$$

Разумеется, эти рецепты можно обобщить и на гораздо более общие случаи.

6.5.1. Чебышевское ускорение

Идея метода заключается в следующем. Напишем параболическое уравнение, итогом эволюции которого будет решение исходной эллиптической задачи:

$$\partial_t u(x, y, t) = \Delta u(x, y, t) + V(u(x, y, t)).$$

Возьмем некоторое начальное приближение $u(x, y, t = 0) = u_0(x, y)$ и будем решать задачу Коши (граничные условия для задачи Коши должны,

разумеется, совпадать с граничными условиями краевой задачи). При этом нам совершенно не нужно заботиться о правильном описании эволюции — при любой, сколь угодно неверной траектории, равновесие наступит только по достижении нужного нам решения, т.е. при правой части, равной нулю. В принципе, возможны два пути: взять неявную схему с достаточно большим dt (см. раздел «Задача Коши для параболических ОДУ»), либо использовать явную схему с таким максимально возможным dt , при котором еще не нарушается условие устойчивости.

Классический метод использует второй путь. Явная схема имеет вид

$$\hat{u}_{mn} = u_{mn} + \frac{dt}{dx^2} [u_{m+1n} + u_{m-1n} + u_{mn+1} + u_{mn-1} - 4u_{mn}] + dtV(u_{mn}).$$

Проводя обычный анализ устойчивости (см. раздел «Задача Коши для гиперболических уравнений»), получаем для амплитуды плоской волны

$$\hat{a} = a \left[1 + \frac{dt}{dx^2} \left(2 \cos(k_x dx) + 2 \cos(k_y dx) - 4 \right) \right].$$

Хуже всего ведут себя волны с $k_x = k_y = \pi/dx$, так что должно быть

$$1 - 8 \frac{dt}{dx^2} > -1,$$

откуда

$$dt \leq \frac{dx^2}{4}.$$

Если мы попытаемся взять dt максимально возможным и будем отслеживать эволюцию системы¹⁴ с помощью схемы:

$$\hat{u}_{mn} = u_{mn} + \frac{1}{4} [u_{m+1n} + u_{m-1n} + u_{mn+1} + u_{mn-1} - 4u_{mn}] + \frac{dx^2}{4} V(u_{mn}),$$

то нас ждет катастрофа.

Заметим прежде всего, что слагаемое $(dx^2/4)V(u_{mn})$ для большинства задач, встречающихся на практике, слегка улучшает ситуацию. Поэтому рассмотрим наихудший случай, т.е. будем полагать это слагаемое равным нулю.

¹⁴Заметим, что выбор $dt = dx^2/4$ означает явное превышение точности при описании эволюции: дифференцирование по времени получается гораздо точнее, чем по координатам.

Предположим, что мы уже почти сошлись к решению. Обозначим разницу между текущими значениями u_{mn} и искомым решением $u_{mn}^{(true)}$ как δu_{mn} . Линеаризуем уравнение относительно δu_{mn} и разложим δu_{mn} по плоским волнам. Все это очень похоже на анализ устойчивости, но анализ устойчивости проводился для небольшого кусочка решетки, а в данном случае мы будем работать на всей решетке сразу. Чтобы разложить δu_{mn} на всей решетке, надо знать граничные условия. Возьмем, к примеру, нулевые граничные условия на периметре прямоугольника $(0 \leq m \leq N) \times (0 \leq n \leq N)$. Другие граничные условия ничего радикально нового не дадут. Тогда волновые числа будут нумероваться целыми индексами p и q :

$$k_x^{(p)} = \pi p / (N dx),$$

$$k_y^{(q)} = \pi q / (N dy).$$

Эволюция всех этих гармоник идет совершенно независимо, причем все они экспоненциально «умирают»:

$$u_{mn}^{(pq)}(t) = a^{(pq)}(t) \sin(\pi p m / N) \sin(\pi q n / N),$$

$$a^{(pq)}(t) = a^{(pq)}(0) \exp(-\lambda_{pq} t),$$

$$\lambda_{pq} = 1 + \frac{1}{2} \left(\cos(\pi p / N) - 1 \right) + \frac{1}{2} \left(\cos(\pi q / N) - 1 \right).$$

Обращение всех гармоник в ноль означает, что мы сошлись к решению. Медленнее всего «умирает» гармоника с $p = q = 1$, так что большая часть времени уходит на то, чтобы амплитуда именно этой гармоники обратилась в ноль. При этом $\lambda_{11} = 1 - \pi^2 / (2N^2) \approx \exp(-\pi^2 / (2N^2))$. Поэтому за M шагов убывание амплитуды этой гармоники составит

$$(\lambda_{11})^M \approx \exp(-M\pi^2 / (2N^2)),$$

т. е. время жизни τ этой гармоники пропорционально N^2 .

На каждом шаге по времени выполняется N^2 операций, количество шагов, необходимых для «смерти» гармоники с $p = q = 1$, пропорционально N^2 , так что всего понадобится N^4 операций. Для любого разумного размера решетки N это совершенно нереально.

Выход из положения поразительно простой. Именно, вместо описания эволюции мы выполняем итерационную процедуру:

$$\hat{u}_{m,n} = u_{m,n} + \frac{\alpha}{4} \left[u_{m+1,n} + \hat{u}_{m-1,n} + u_{m,n+1} + \hat{u}_{m,n-1} - 4u_{m,n} \right] + \frac{dx^2}{4} V(u_{m,n}).$$

Несмотря на наличие $\hat{u}_{m,n}$ в правой части, это явная процедура. Дело в том, что к моменту вычисления $\hat{u}_{m,n}$ значения $\hat{u}_{m-1,n}$ и $\hat{u}_{m,n-1}$ уже известны.¹⁵ Выясним, при каких α процедура устойчива. Положим $k_x = k_y = k$ и выясним, как меняется амплитуда плоской волны за один шаг:

$$\hat{a}/a = \frac{1 + (\alpha/2)[\exp(i k dx) - 2]}{1 - (\alpha/2)\exp(-i k dx)},$$

отсюда

$$|\hat{a}/a|^2 - 1 = \frac{\alpha(\alpha - 2)(1 - \cos(k dx))}{1 + \alpha^2/4 - \alpha \cos(k dx)}.$$

Видно, что устойчивость достигается при $\alpha \leq 2$. Положим теперь $\alpha = 2 - 2\epsilon$ и выясним, чему равно время жизни самой медленной гармоники $k_x = k_y = \pi/(N dx)$. Для этой гармоники $\cos(k dx) = \cos(\pi/N)$. Введем обозначение $\delta \equiv \pi/N$ и разложим $\cos(\delta)$ по малому аргументу δ . Тогда получим:

$$|\hat{a}/a|^2 - 1 \approx -\frac{2\epsilon\delta^2}{\epsilon^2 + \delta^2}.$$

Легко проверить, что минимум выражения достигается при $\epsilon = \delta$, при этом $|\hat{a}/a|^2 - 1 = -\delta$. Тогда $|\hat{a}/a| \approx 1 - \delta/2$, т. е. время жизни τ самой медленной гармоники пропорционально $1/\delta$.

Итак, $\tau \sim 1/\delta \sim N^1$ вместо прежнего $\tau \sim N^2$. Следовательно, за счет небольших изменений в алгоритме мы уменьшили количество операций в N раз: N^3 операций вместо N^4 операций. Заметим еще, что особой точности в определении ϵ не нужно. Если взять любое $\epsilon = \beta\delta$, то время жизни останется пропорциональным N^1 , но умножится на $(\beta + 1/\beta)/2$. Это нежелательно, но не катастрофично.

Однако это еще не окончательный рецепт.

Значение $\alpha = 2 - 2\delta$ означает устойчивость, но не означает быстрое гашение высокочастотных гармоник. Действительно, при $k = \pi/dx$ получаем

$$\hat{a}/a = \frac{1 - (3/2)\alpha}{1 + (1/2)\alpha} \approx -1 + \delta/2.$$

Итак, при $\alpha = 2 - 2\delta$ высокочастотные гармоники на каждом шаге меняют знак, а их амплитуда убывает так же медленно, как и у самой низкочастотной гармоники. Положение можно исправить, если первые шаги делать

¹⁵Если Вы читали про алгоритм Гаусса-Зейделя итерационного решения СЛУ, то происхождение этой процедуры для Вас должно быть совершенно очевидным.

при $\alpha \approx 1$, и только потом постепенно увеличивать α , устремляя ее к предельному значению $\alpha = 2 - 2\delta$. Заметим, что для каждой частоты есть своя оптимальная α , при которой убывание идет быстрее всего. Например, для гармоник с волновым числом $k = \pi/dx$ оптимальное значение составляет $\alpha = 2/3$: при этом амплитуда обращается в ноль за один шаг. Поэтому если мы будем постепенно увеличивать α от 1 до $2 - 2\delta$, то мы попадем (пусть и ненадолго) в области оптимальных значений α для каждой из частот. Практически предлагается на s -ом шаге процедуры использовать $\alpha_s = 2 - 2\delta - 1/s$. Идея использовать переменное α как раз и называется чебышевским ускорением. Как показывает опыт непосредственных вычислений, использование переменного α действительно заметно ускоряет сходимость.

Изложенная процедура — это один из рекомендуемых методов решения краевой задачи для ДУЧП. Следует только помнить, что в существенно нелинейных задачах наша оценка для оптимального значения $\alpha = 2 - 2\delta$ может слегка сдвинуться, так что правильное значение оптимального α надо будет подбирать эмпирически.

6.5.2. Минимизация

В отличие от одномерных задач, т. е. от случая ОДУ, метод минимизации оказывается вполне сопоставимым по скорости с классическим методом, изложенным в предыдущем разделе. Дело в том, что поиск минимума невязки динамическим методом (см. часть I, раздел «Многомерные минимумы») есть в точности решение задачи Коши для гиперболического уравнения с трением, а гиперболическое уравнение накладывает менее жесткое ограничение на dt , чем параболическое, использованное в предыдущем разделе.

Итак, мы используем следующую процедуру:

$$\begin{aligned} \hat{v}_{mn} = \beta \left[v_{mn} + \frac{dt}{dx^2} \left[u_{m+1n} + u_{m-1n} + u_{mn+1} + u_{mn-1} - 4u_{mn} \right] + \right. \\ \left. + dtV(u_{mn}) \right] \equiv \beta(v_{mn} + dt f_{mn}), \end{aligned}$$

$$\hat{u}_{mn} = u_{mn} + dt \hat{v}_{mn}.$$

Заметим, что «силы» f_{mn} , действующие на величины u_{mn} , есть в точности невязка нашего исходного эллиптического уравнения. Достижение точки равновесия означает обращение всех сил f_{mn} в ноль, следовательно,

в точке равновесия величины u_{mn} будут решением нашего эллиптического уравнения. Подчеркнем, что в данном случае мы, строго говоря, минимизируем вовсе не невязку. Мы минимизируем тот функционал, градиенты которого совпадают с невязкой. Понятно, что минимум такого функционала соответствует нулевой невязке, т. е. решению исходного уравнения.

Как обычно, недостатком динамического метода является необходимость подгонки параметров dt и β под конкретную задачу. В принципе dt должно быть порядка dx , а β — в пределах $0.7 \dots 0.999$.

Кроме того, сходимость можно ускорить, если параметр β сделать переменным. Для этого надо вычислять косинус угла между скоростью v_{mn} и силой f_{mn} :

$$\cos(\theta) = \frac{\sum_{mn} f_{mn} v_{mn}}{\sqrt{\sum_{mn} v_{mn}^2} \sqrt{\sum_{mn} f_{mn}^2}}.$$

Тогда при $\cos(\theta) < 0.5$ следует полагать $\beta = 0.5$, а в противном случае брать либо $\beta = \cos(\theta)$, либо, что еще лучше, $\beta = [\cos(\theta)]^\gamma$, где $\gamma = 0.5 \dots 0.001$. Опять-таки, параметр γ приходится подстраивать под конкретную задачу, что является недостатком метода.

Изложенная процедура — это второй из рекомендуемых методов решения краевой задачи для ДУЧП. В существенно нелинейных задачах этот метод нередко оказывается заметно эффективнее классического чебышевского ускорения.

6.5.3. Мультирешетки

Основная идея мультирешеточных методов довольно изящна. Обычно нам не требуется знание величин $u(x, y)$ на очень частой решетке, нам хватило бы и довольно грубой решетки. Уменьшать шаг, т. е. брать более частую решетку, приходится в основном ради того, чтобы получить достаточно точное решение. Между тем маленький шаг означает большое количество узлов N и очень медленную сходимость.

При этом не следует думать, что более частая решетка позволяет лишь уточнить детали профиля решения, т. е. корректирует высокочастотные компоненты решения. Нередко использование более частой решетки дает заметные сдвиги и в низкочастотных компонентах решения. Происходит это из-за взаимного влияния высоко- и низкочастотных гармоник в нелинейных задачах. Тем не менее общее представление о профиле решения (т. е. о низкочастотных компонентах решения) на грубой решетке получить можно, пусть и не очень точно. Как раз эти низкочастотные компоненты решения

медленнее всего сходятся на более частой решетке (количество операций для них порядка N^3 , т. е. порядка $1/dx^3$).

Отсюда возникает идея: получить приблизительно правильный профиль на грубой решетке, а уж потом использовать его в качестве начального приближения на более частой решетке, и, тем самым, уточнить. Процедура уточнения займет меньше времени, чем полная процедура поиска решения.

При этом более частая решетка должна содержать исходную решетку как подмножество. Простейший путь — уменьшить dx в два раза, т. е. добавить к исходной решетке новые узлы, расположенные в серединках ребер и серединках квадратиков старой решетки. При этом значения $u_{n,m}$ в новых (добавленных) узлах получаются с помощью интерполяции (см. часть I, раздел «Интерполяция»). Заметим, что ограничиваться билинейной интерполяцией было бы весьма неразумно. Достаточно удовлетворительный результат дает последовательная интерполяция по четырем соседним узлам. Именно, сначала вычисляются значения функции в новых узлах $u_{n+1/2,m}$, причем величина $u_{1/2,m}$ вычисляется с помощью полиномиальной интерполяции по значениям $u_{0,m}$, $u_{1,m}$, $u_{2,m}$, $u_{3,m}$; величина $u_{N_x-1/2,m}$ — по значениям $u_{N_x,m}$, $u_{N_x-1,m}$, $u_{N_x-2,m}$, $u_{N_x-3,m}$; и, наконец, величина $u_{n+1/2,m}$ для $n = 1, \dots, N_x - 2$ — по значениям $u_{n-1,m}$, $u_{n,m}$, $u_{n+1,m}$, $u_{n+2,m}$. После чего точно таким же образом находятся величины $u_{n,m+1/2}$ (с помощью $u_{n,m}$) и величины $u_{n+1/2,m+1/2}$ (с помощью ранее найденных $u_{n+1/2,m}$).

Более того, если процедуру измельчения решетки проводить несколько раз подряд, то напрашивается применение интерполяционных методов (см. часть I, раздел «Численное интегрирование. Алгоритм Ромберга»). Действительно, мы располагаем значениями $u_{n,m}^{(1)}$ на самой первой (самой грубой) решетке с шагом $dx^{(1)}$; уточненными значениями $u_{n,m}^{(2)}$ в тех же узлах, но вычисленными с помощью более частой решетки ($dx^{(2)} = dx^{(1)}/2$), располагаем еще более точными значениями $u_{n,m}^{(3)}$ в тех же исходных узлах, но на решетке с $dx^{(3)} = dx^{(1)}/4$, и т. д. Следовательно, у нас есть не просто текущее значение $u_{n,m}$ на самой тонкой из использованных решеток, а зависимость этого $u_{n,m}$ от шага dx . Разумеется, эту зависимость следует проинтерполировать в точку $dx^{(\infty)} = 0$. Как обычно, в качестве аргумента функции $Y(X)$ следует брать не сам шаг dx , а его квадрат: $Y(X) = u_{n,m}(dx^2)$.

Авторы настоятельно рекомендуют этот «последовательный» алгоритм измельчения решетки с завершающей интерполяцией. Заметим, что мы ни разу не уточнили, каким именно методом мы обрабатываем каждую из решеток. В данном случае это совершенно неважно.

Кроме «последовательного» алгоритма существует еще и «параллельный».

При применении «параллельного» алгоритма мы изначально работаем с самой частой решеткой, во всяком случае невязку мы вычисляем именно на ней. Затем мы «поднимаемся» до самой грубой решетки, т. е., пользуясь значениями $u_{n,m}$ и невязки на самой частой решетке, получаем значения $u_{n,m}$ и невязку на самой грубой. На этой грубой решетке мы делаем один (или несколько) шагов (это позволяет достаточно быстро сдвинуться в правильную сторону по низкочастотным гармоникам). Затем мы опять «спускаемся» вниз до самой частой решетки.

При этом на каждой из промежуточных решеток обязательно делается один или несколько сглаживающих шагов, аналогичных начальным шагам в методе чебышевского ускорения (см. раздел «Чебышевское ускорение»). Иными словами, мы берем $\alpha = 1$ вместо $\alpha = 2 - 2\delta$. Это позволяет устранить ошибки в высокочастотных гармониках, которые с неизбежностью возникают при переходе от более грубой решетки к более частой.

Затем на самой частой решетке мы вновь вычисляем невязку и т. д.

В «параллельном» алгоритме низкочастотные гармоники сдвигаются на самой грубой решетке, но невязка для них вычисляется правильная, поскольку она вычисляется на самой частой решетке.

Вариантов движения по «лесенке» из решеток существует очень много. Необязательно каждый раз подниматься до самой грубой решетки, а потом спускаться до самой частой. Например, можно один раз подняться до середины лесенки, потом опять спуститься вниз, а уж второй раз подниматься до самого верха «лесенки». Такого рода рецепты можно придумывать в неограниченном количестве.

При этом спуск на одну ступеньку вниз (к более частой решетке) предлагается делать совершенно тривиальным образом — билинейной интерполяцией.¹⁶

$$u_{n+1/2,m}^{(k+1)} = (1/2)u_{n,m}^{(k)} + (1/2)u_{n+1,m}^{(k)},$$

$$u_{n,m+1/2}^{(k+1)} = (1/2)u_{n,m}^{(k)} + (1/2)u_{n,m+1}^{(k)},$$

$$u_{n+1/2,m+1/2}^{(k+1)} = (1/4)u_{n,m}^{(k)} + (1/4)u_{n+1,m}^{(k)} + (1/4)u_{n,m+1}^{(k)} + (1/4)u_{n+1,m+1}^{(k)}.$$

А вот подъем на одну ступеньку вверх (как для значений $u_{n,m}$, так и для невязки) предлагается делать вовсе не механическим отбрасыванием

¹⁶На наш взгляд, весьма неудачный выбор.

лишних узлов. Мы вовсе не полагаем, что $u_{n,m}^{(k)} = u_{n,m}^{(k+1)}$. Оказывается, что необходимо учитывать величины в соседних узлах:

$$u_{n,m}^{(k)} = \frac{1}{4}u_{n,m}^{(k+1)} + \frac{1}{8} \left[u_{n+1/2,m}^{(k+1)} + u_{n-1/2,m}^{(k+1)} + u_{n,m+1/2}^{(k+1)} + u_{n,m-1/2}^{(k+1)} \right] + \\ + \frac{1}{16} \left[u_{n+1/2,m+1/2}^{(k+1)} + u_{n-1/2,m+1/2}^{(k+1)} + u_{n+1/2,m-1/2}^{(k+1)} + u_{n-1/2,m-1/2}^{(k+1)} \right].$$

Этот довольно неожиданный рецепт используется ради сохранения скалярного произведения (и, следовательно, нормы) при перемещении по «лесенке» решеток.

Именно, пусть есть две функции: $u_{n,m}^{(k)}$, определенная на более грубой k -й решетке, и $v_{n,m}^{(k+1)}$, определенная на более частой $(k+1)$ -й решетке. Потребуем, чтобы их скалярное произведение, вычисленное на k -й решетке, совпадало со скалярным произведением, вычисленным на $(k+1)$ -й решетке:

$$h_k^2 \sum_{m,n} u_{n,m}^{(k)} v_{n,m}^{(k)} = h_{k+1}^2 \sum_{a,b} u_{a,b}^{(k+1)} v_{a,b}^{(k+1)}.$$

При этом $h_{k+1} = h_k/2$, а индексы a, b мы будем полагать полуцелыми, т. е. либо $a = m$, либо $a = m + 1/2$.

Если $u_{a,b}^{(k+1)}$ получается билинейной интерполяцией, то на $(k+1)$ -й решетке получаем:

$$\frac{h_k^2}{4} \sum \left[u_{n,m}^{(k)} v_{n,m}^{(k+1)} + \frac{1}{2} \left(u_{n,m}^{(k)} + u_{n+1,m}^{(k)} \right) v_{n+1/2,m}^{(k+1)} + \right. \\ \left. + \frac{1}{2} \left(u_{n,m}^{(k)} + u_{n,m+1}^{(k)} \right) v_{n,m+1/2}^{(k+1)} + \right. \\ \left. + \frac{1}{4} \left(u_{n,m}^{(k)} + u_{n+1,m}^{(k)} + u_{n,m+1}^{(k)} + u_{n+1,m+1}^{(k)} \right) v_{n+1/2,m+1/2}^{(k+1)} \right].$$

Перегруппировав слагаемые, получим:

$$h_k^2 \sum u_{n,m}^{(k)} \left\{ \frac{1}{4} v_{n,m}^{(k+1)} + \frac{1}{8} \left[v_{n+1/2,m}^{(k+1)} + v_{n-1/2,m}^{(k+1)} + v_{n,m+1/2}^{(k+1)} + v_{n,m-1/2}^{(k+1)} \right] + \right. \\ \left. + \frac{1}{16} \left[v_{n+1/2,m+1/2}^{(k+1)} + v_{n-1/2,m+1/2}^{(k+1)} + v_{n+1/2,m-1/2}^{(k+1)} + v_{n-1/2,m-1/2}^{(k+1)} \right] \right\}.$$

Если мы требуем сохранения скалярного произведения, то коэффициент при $u_{n,m}^{(k)}$, взятый в фигурные скобки, должен совпадать с $v_{n,m}^{(k)}$.

Так что инвариантность скалярного произведения действительно позволяет получить вполне однозначный (и довольно нетривиальный) рецепт перехода $u_{n,m}^{(k+1)} \rightarrow u_{n,m}^{(k)}$.

Поскольку при написании «параллельного» алгоритма «возни» гораздо больше, чем при написании «последовательного», а заметного выигрыша в скорости он обычно не дает, нам кажется, что лучше его не использовать. Тем не менее, окончательное решение мы оставляем за читателем.

7. Интегральные уравнения. Нелокальные уравнения

В этом разделе мы (как обычно, довольно поверхностно) обсудим численное решение интегральных уравнений и уравнений, которые к ним сводятся. Известно, что большинство нелокальных уравнений приходится сводить к интегральным уравнениям. Рассмотрим, например, конечно-разностное уравнение с мнимым шагом:¹⁷

$$-\left(\psi(x+i) + \psi(x-i) - 2\psi(x)\right) + (E - V(x))\psi(x) = 0 \quad (*)$$

Разумеется, его можно решать и непосредственно: ввести сетку x_k на вещественной оси, а значения $\psi(x_k \pm i)$ получать, интерполируя (точнее, экстраполируя) функцию $\psi(x)$ в точки на комплексной плоскости. Однако, можно получить гораздо более точные численные результаты (и даже приближенное аналитическое решение), если превратить это уравнение в интегральное. Рецепт такого превращения хорошо известен. Достаточно вспомнить, как в квантовой теории рассеяния уравнение Шредингера заменяют на интегральное уравнение Липпмана-Швингера.

Сначала мы выписываем уравнение для функции Грина, соответствующей (*):

$$-\left(g(k, x+i) + g(k, x-i) - 2g(k, x)\right) + \\ + \left(2 \cos k - 2\right)g(k, x) = \delta(x),$$

¹⁷Это спектральное уравнение возникает в теории поля при релятивистски-инвариантном описании возмущений нетривиального классического решения, если рассматривать состояния системы, лежащие на массовой поверхности.

где $E = 2 \cos k - 2$. Затем находим саму функцию Грина:

$$g(k, x) = -\frac{1}{\sin k} \frac{\operatorname{sh}((\pi - k)x)}{\operatorname{sh}((\pi - 0)x)}.$$

После этого выписываем интегральное уравнение:

$$\psi(x) = \int dy g(k, x - y) V(y) \psi(y).$$

Решив это интегральное уравнение, мы получим гораздо более точный ответ, чем при использовании интерполяции.

Другой пример нелокального уравнения — уравнение с бесконечным количеством производных:

$$\exp(\partial_x^2) \phi(x) + V(\phi(x)) = 0,$$

которое обыкновенно возникает в струнных теориях. Здесь уже никакая интерполяция не поможет, это уравнение необходимо превращать в интегральное.

Простейшие методы решения интегральных уравнений заключаются в том, что мы вводим ту или иную решетку x_k , а затем так или иначе изображаем интеграл в виде суммы по узлам решетки. В результате исходное интегральное уравнение сводится к системе линейных уравнений (СЛУ).

Иногда целесообразно не решать задачу непосредственно в x -пространстве, а разложить функцию по какому-либо базису (в ряд Фурье, по классическим ортогональным полиномам и т. д.). После этого можно выписать СЛУ для коэффициентов разложения по этому базису. Например, для трансляционно-инвариантных ядер, для которых выполняется условие $K(x, y) = K(x - y)$, напрашивается разложение в ряд Фурье.

Кстати, если уравнение нелинейное, то мы опять-таки можем ввести решетку и изобразить интеграл в виде суммы по узлам решетки. После этого придется решать получившуюся нелинейную систему уравнений (см. часть I, раздел «Многомерные корни»).

Заметим, наконец, что уравнение Фредгольма I рода:

$$\int dy K(x, y) f(y) = g(x)$$

вынесено за рамки этого раздела. Оно обсуждается в следующем разделе, посвященном некорректно поставленным задачам, поскольку метод его решения совершенно не похож на методы решения, излагаемые в этом разделе.

7.1. Уравнение Вольтерра

Это самый симпатичный класс интегральных уравнений. Вероятно, именно поэтому он так редко встречается на практике. Уравнение имеет вид:

$$f(x) = \int_a^x dy K(x, y) f(y) + g(x), \quad x \in [a, b].$$

Интегральный оператор в данном случае имеет треугольное ядро. Следовательно, матрица той системы линейных уравнений, которую мы получим, тоже будет треугольной. Количество операций при решении СЛУ размера $N \times N$ с треугольной матрицей пропорционально N^2 (а не N^3 , как для СЛУ общего вида). Так что мы можем позволить себе достаточно частую сетку и достаточно мелкий шаг. Итак, берем равномерную сетку $x_k = a + k dx$, $k = 0 \dots N$, $dx = (b - a)/N$, $f(x_k) \equiv f_k$. Используем формулу трапеций. Тогда для первого узла:

$$f_0 = g_0,$$

для второго узла:

$$f_1 = g_1 + dx \left[\frac{1}{2} K(x_1, x_0) f_0 + \frac{1}{2} K(x_1, x_1) f_1 \right]$$

и, наконец, для n -го узла ($2 \leq n \leq N$):

$$f_n = g_n + dx \left[\frac{1}{2} K(x_n, x_0) f_0 + \sum_{k=1}^{n-1} K(x_n, x_k) f_k + \frac{1}{2} K(x_n, x_n) f_n \right].$$

Так что из первого уравнения находим f_0 , из второго — f_1 и т. д., из $N + 1$ -го уравнения находим f_N .

Возникает естественный вопрос, почему мы взяли самую примитивную интегральную схему, а не, скажем, метод Симпсона. Ответ приблизительно такой же, как и для численного интегрирования. Напомним, что правильный алгоритм вычисления определенного интеграла — это не метод Симпсона, а интерполяционный алгоритм Ромберга.

Действительно, пусть мы хотим получить значения искомой функции f_k в узлах решетки с шагом dx_1 . Решаем треугольную систему и получаем ответ $f_k(dx_1)$. Но мы должны еще и проконтролировать точность полученного ответа. В данном случае проконтролировать точность — это использовать новую сетку с удвоенным количеством узлов и шагом $dx_2 = dx_1/2$ (работы будет в 4 раза больше). Получаем ответ $f_k(dx_2)$ (в том числе и в тех промежуточных точках, которые нам не нужны). Сравниваем ответы для

тех узлов, которые являются общими для старой и новой сетки, т.е. для всех узлов старой сетки. Если погрешность не превышает заказанной, то можно число узлов больше не удваивать. Но ответом f_k следует считать не $f_k(dx_2)$, а результат полиномиальной интерполяции зависимости $f_k(dx)$ в точку $dx = 0$.¹⁸ Полиномиальная интерполяция по двум точкам как раз и даст формулу Симпсона. Если погрешность велика, то процедуру удвоения числа узлов надо продолжать, при этом интерполяция пойдет уже по трем (и более) точкам.

7.2. Задача на собственные значения

Как известно, задача на собственные значения (СЗ) интегрального оператора пишется «наоборот»:

$$f(x) = \lambda \int_a^b dy K(x, y) f(y), \quad x \in [a, b],$$

т.е. собственное значение является множителем при операторе. Разумеется, мы должны так или иначе заменить интеграл на интегральную сумму и искать СЗ получившейся матрицы. Чем меньше размер матрицы, тем проще эта задача, так что есть необходимость бороться за минимизацию числа узлов решетки. Как известно, абсолютным «чемпионом» по точности при данном числе узлов решетки N является метод Гаусса (см. часть I, раздел «Численное интегрирование»). Напомним, что в этом методе сначала выбирается то или иное семейство классических ортогональных полиномов (КОП), определенных на интервале $[a, b]$ с весом $\rho(x)$ (обычно выбирают полиномы Лежандра или Чебышева). Затем фиксируют число N — это будет число узлов решетки. Затем находят все корни N -го полинома x_k , $k = 1 \dots N$. Затем решают СЛУ для весов w_k :

$$\sum_{k=1}^N \rho(x_k) P_l(x_k) w_k = \delta_{l,0} \int_a^b dx \rho(x), \quad l = 0 \dots N-1.$$

После этого интеграл от любой функции $f(x)$ по интервалу $[a, b]$ может быть записан в виде

$$\int_a^b dx f(x) \approx \sum_{k=1}^N f(x_k) w_k.$$

¹⁸Полезно напомнить, что аргументом функции $Y(X)$ следует считать не сам шаг dx , а его квадрат dx^2 : $Y(X) = f_k(dx^2)$ (см. часть I, раздел «Численное интегрирование. Алгоритм Ромберга»).

Эта формула дает совершенно точный ответ для $(2N - 1)$ штук эталонных функций $\phi_l(x)$ вида $\phi_l(x) = \rho(x)P_l(x)$. Иными словами, это квадратурная формула $(2N - 1)$ -го порядка точности. Разумеется, мы проделали лишнюю работу по поиску всех корней $P_N(x)$ и решению СЛУ для весов w_k . Но в данном случае, когда необходимо минимизировать число узлов N , не ухудшая точности вычисления, эта лишняя работа вполне окупается. Следует заметить, что как корни, так и веса для данного семейства КОП и для данного N вычисляются один раз и навсегда.

Итак, мы заменяем интеграл на интегральную сумму. В результате получается обычная задача на СЗ квадратной матрицы A_{ij} :

$$\sum_{j=1}^N K_{ij} w_j f_j \equiv \sum_{j=1}^N A_{ij} f_j = \lambda^{-1} f_i, \quad (*)$$

где $K_{ij} \equiv K(x_i, x_j)$, $f_i \equiv f(x_i)$. Если ядро изначально неэрмитово, т. е. если $K(x, y) \neq K(y, x)$, то вполне можно примириться с неэрмитовостью матрицы A_{ij} . Однако, если исходная задача была эрмитовой, т. е. $K(x, y) = K(y, x)$, то, написав уравнение (*), мы эту эрмитовость разрушили. Действительно, матрица в уравнении (*) явно неэрмитова:

$$A_{ij} = K_{ij} w_j \neq A_{ji} = K_{ji} w_i,$$

хотя $K_{ij} = K_{ji}$. К счастью, эрмитовость легко восстановить, если положить $f_i \equiv y_i / \sqrt{w_i}$. Тогда мы получим задачу на СЗ для эрмитовой матрицы $\tilde{A}_{ij}$:

$$\sum_{j=1}^N \sqrt{w_i} K_{ij} \sqrt{w_j} y_j \equiv \sum_{j=1}^N \tilde{A}_{ij} y_j = \lambda^{-1} y_i.$$

Методы решения задачи на СЗ квадратной матрицы хорошо известны (см. часть I, раздел «Задача на СВ и СЗ»).

7.3. Уравнение Фредгольма II рода

Уравнение имеет вид:

$$f(x) = \int_a^b dy K(x, y) f(y) + g(x), \quad x \in [a, b].$$

Как известно, если среди СЗ интегрального оператора (см. предыдущий раздел) нет собственного значения, равного единице, то это уравнение

имеет единственное решение. Это утверждение легко перевести на язык численного счета. Действительно, очевидно, что, как и при поиске СЗ интегрального оператора, в этой задаче необходимо минимизировать число узлов решетки N , ведь потом придется решать СЛУ размерности $N \times N$. Поэтому снова придется использовать метод Гаусса. Заменяем интеграл на интегральную сумму:

$$f_i = \sum_{j=1}^N K_{ij} w_j f_j + g_i$$

или

$$\sum_{j=1}^N (\delta_{ij} - A_{ij}) f_j = g_i. \quad (*)$$

Так что если среди СЗ матрицы A_{ij} нет единицы, то матрица нашей СЛУ несингулярна, и у СЛУ (*) существует единственное решение.

Методы решения СЛУ хорошо известны (см. часть I, раздел «Решение СЛУ»).

Последнее замечание, которое необходимо сделать, касается вычисления функции $f(x)$ в промежуточных точках. Решение СЛУ (*) дает нам значения функции в узлах решетки $f(x_k) = f_k$. Как ни странно, оптимальные результаты при вычислении $f(x)$ в промежуточных точках получаются вовсе не при использовании того или иного интерполяционного метода, а при вычислении величины:

$$f(x) \approx \sum_{j=1}^N K(x, x_j) w_j f_j + g(x).$$

7.4. Итерационный метод

Итерационный метод — это своего рода метод «стрельбы» для интегральных уравнений. Он далеко не всегда работает, но если он работает, то применять следует именно его. Идея метода совершенно тривиальна. Мы берем некоторое начальное приближение к решению $f^{(0)}(x)$. А затем устраиваем рекурсивную процедуру:

$$f^{(k+1)}(x) = \int_a^b dy K(x, y) f^{(k)}(y) + g(x).$$

Нередко уже за 5–6 итераций процедура сходится, т. е. достигается равенство $f^{(k+1)}(x) = f^{(k)}(x)$. Поскольку здесь не требуется решать СЛУ, то нет особой необходимости использовать Гауссовы квадратуры. Вполне можно

использовать рецепт, изложенный в разделе «Уравнение Вольтерра», с методом трапеций, делением шага пополам и последующей интерполяцией. Впрочем, для уравнения Вольтерра мы за N^2 операций получим окончательный ответ, а здесь мы за N^2 операций сделаем всего лишь один шаг итерационной процедуры. Однако, если сходимость будет достигнута, скажем, за 10 шагов, то это все равно гораздо лучше, чем N^3 операций, нужных для решения СЛУ.

8. Некорректные задачи

Простейшим примером некорректной задачи является решение интегрального уравнения Фредгольма I рода:

$$\int_a^b dy K(x, y) f(y) = g(x), \quad x \in [a, b].$$

Как и уравнение Фредгольма II рода, оно с легкостью превращается в СЛУ (см. предыдущий раздел). Единственное (но фатальное) отличие заключается в том, что из матрицы СЛУ исчезает добавок, равный единичной матрице:

$$\sum_{j=1}^N K_{ij} w_j f_j \equiv \sum_{j=1}^N A_{ij} f_j = g_i.$$

Из-за этого матрица полученной СЛУ A_{ij} оказывается сингулярной. Напомним, что с точки зрения численного счета сингулярность означает вовсе не нулевой (или очень маленький) детерминант. Матрица сингулярна, если среди семейства ее собственных значений (СЗ) λ_n есть группа СЗ, которые можно считать равными нулю по сравнению с остальными СЗ. Другими словами, матрица сингулярна, если есть хотя бы одна пара СЗ, для которых отношение $|\lambda_n/\lambda_m|$ меньше машинного нуля. Действительно, пусть у матрицы есть собственные векторы (СВ) ψ_n с собственными значениями λ_n порядка единицы и, кроме того, несколько «нерегулярных» собственных векторов с собственными значениями порядка машинного нуля. Тогда ко всякому решению можно безболезненно добавить любую линейную комбинацию этих «нерегулярных» собственных векторов, при этом решение нисколько не испортится. Более того, поскольку спектральное разложение обратной матрицы имеет вид $\sum_n |\psi_n\rangle \lambda_n^{-1} \langle \psi_n|$, то при любом стандартном алгоритме решения СЛУ коэффициенты при «нерегулярных» СВ будут гораздо больше, чем коэффициенты при остальных СВ. Кроме того, эти ко-

эффективности будут разными при применении разных алгоритмов решения СЛУ.

Сингулярность матрицы A_{ij} заранее очевидна. Действительно, интеграл от любой быстро осциллирующей функции близок к нулю, так что у матрицы A_{ij} заведомо есть не один, а много СВ с практически нулевыми СЗ. Для уравнения Фредгольма II рода это не фатально. Действительно, для уравнения Фредгольма II рода матрица СЛУ имеет вид $\delta_{ij} - A_{ij}$, так что все эти почти нулевые СЗ превращаются в почти единичные СЗ, а единичные СЗ никак не мешают решению СЛУ.

Для уравнения Фредгольма I рода добавок δ_{ij} отсутствует, так что мы решаем сингулярную СЛУ. Применяя любой из алгоритмов решения СЛУ¹⁹ мы получим такой ответ, который совершенно не напоминает гладкую функцию. Он будет иметь разные знаки в соседних узлах решетки, а его амплитуда будет гораздо больше, чем у «истинного» гладкого решения.

Довольно ясно, что практически любая обратная задача в физике с неизбежностью приводит к некорректной задаче. Обратная задача электростатики, обратная задача магнитостатики, ... , заведомо будут иметь семейство решений, в которых заряды (токи, ...) в соседних узлах решетки практически компенсируют друг друга, т.е. имеют разные знаки при несуразно больших абсолютных величинах.

Подчеркнем, что дело не в неточном решении СЛУ, а в том, что полученное точное решение не удовлетворяет пусть и естественным, но дополнительным по отношению к исходной постановке задачи условиям — условиям гладкости. Например, при решении задачи «туда-обратно», т.е. если сначала взять гладкое распределение зарядов, затем вычислить поля по зарядам, а потом снова найти заряды по полям, решая соответствующую СЛУ, то в результате мы с неплохой точностью получим исходное гладкое распределение зарядов.

Разумеется, некорректная задача не обязана быть линейной, но мы рассмотрим линейный случай, т.е. задачу $\hat{A}\vec{x} = \vec{b}$ для «численно сингулярной» матрицы $\hat{A}$. Обычно рекомендованные ниже рецепты можно применять и в нелинейном случае.

8.1. «Примитивный» рецепт

Как ни странно, можно просто решать СЛУ методом минимизации невязки, т.е. находя минимум «функционала» $(\hat{A}\vec{x} - \vec{b})^2$. При этом не вво-

¹⁹Кроме того алгоритма, в котором мы сначала ищем СВ и СЗ матрицы A_{ij} ; затем строим спектральное разложение обратной матрицы $\sum_n |\psi_n\rangle \lambda_n^{-1} \langle \psi_n|$; а затем выкидываем из этого разложения все слагаемые, соответствующие «нерегулярным» почти нулевым СЗ.

дится никаких дополнительных условий (никакого функционала регуляризации). Это особенно подходящий способ, когда матрица $\hat{A}$ настолько велика, что даже не помещается в памяти, т. е. прямое решение СЛУ (триангуляция или LU-разложение) нереально. Если каждый отдельный элемент матрицы A_{ij} вычисляется сравнительно быстро, то при вычислении градиента функции невязки (т. е. величины $\hat{A}^T[\hat{A}\vec{x} - \vec{b}]$) проблем не возникает — сначала вычисляются $\vec{y} = [\hat{A}\vec{x} - \vec{b}]$ (это потребует всего N^2 операций), а уж потом $\hat{A}^T\vec{y}$ (опять-таки N^2 операций).

Разумеется, в качестве начального приближения следует взять нечто гладкое. При поиске минимума динамическим методом нет никаких причин, которые увели бы нас слишком далеко от подпространства гладких функций. Иными словами, мы будем двигаться в подпространстве сравнительно гладких решений в сторону минимума функции $(\hat{A}\vec{x} - \vec{b})^2$ и рано или поздно туда попадем (напомним, что динамический метод сходится к точке равновесия, т. е. к точке $\hat{A}\vec{x} - \vec{b} = 0$.)

8.2. Регуляризация

Следует подчеркнуть, что некорректным задачам, регуляризации, решению обратных задач посвящены целые монографии. Мы приведем лишь простейший рецепт, который, впрочем, оказался вполне удовлетворительным в тех задачах, которые попадались авторам.

Для регуляризации задачи необходимо так или иначе формализовать критерий приемлемости решения. Разумеется, речь может идти о любом условии, которому должно удовлетворять решение, не только о гладкости функции. Впрочем, условие гладкости встречается чаще всего. Оптимальный способ реализации условия гладкости зависит от конкретной задачи. Есть задачи, где следует запрещать резкие скачки функции от одного узла решетки к другому. Это можно сделать, определив «функционал» $\sum_i (f_{i+1} - f_i)^2$. Относительно малым значениям этого функционала соответствуют достаточно гладкие решения. Однако, этот функционал запрещает еще и линейные функции с достаточно большим наклоном. Поэтому есть задачи, где лучше использовать функционал $\sum_i (f_{i+1} - 2f_i + f_{i-1})^2$. Этот функционал равен нулю для любых линейных функций. Иногда имеет смысл использовать линейную комбинацию этих функционалов.

Нередко имеет смысл перейти к Фурье-образам f_k наших функций. В этом случае условие гладкости эквивалентно условию запрета на все высокочастотные гармоники. Впрочем, как мы знаем (см. часть I, разделы «Фурье-интерполяция» и «Быстрое преобразование Фурье»), совсем за-

нулять высокочастотные гармоники нельзя, их следует именно подавить, причем тем сильнее, чем больше частота (волновое число). Так что в качестве функционала надо брать $\sum |f_k|^2 F(k)$, где функция подавления $F(k)$ почти равна нулю для низкочастотных гармоник и достаточно велика для высокочастотных.

Если задача линейна, то и функционал дополнительного условия имеет смысл выбирать линейным, т. е. представлять его в виде $(\hat{R}\vec{x} - \vec{r})^2$. Величины $\vec{r}$ для большинства условий, встречающихся на практике, равны нулю.

А теперь, чтобы найти приближенное решение нашего линейного уравнения, которое еще к тому же будет удовлетворять дополнительному условию (например, условию регулярности), мы должны найти минимум функционала

$$\Phi(\vec{x}) = (\hat{A}\vec{x} - \vec{b})^2 + \epsilon(\hat{R}\vec{x})^2,$$

где ϵ — некоторая константа. Если размер матрицы позволяет, то можно попросту решать СЛУ:

$$[\hat{A}^T \hat{A} + \epsilon \hat{R}^T \hat{R}] \vec{x} = \hat{A}^T \vec{b},$$

в противном случае следует искать минимум функционала $\Phi(\vec{x})$ с помощью стандартных алгоритмов поиска минимума.

Очевидно, что ответ будет зависеть от константы ϵ . Возникает естественный вопрос, как ее выбирать. Универсального ответа не существует. В принципе существуют алгоритмы автоматического выбора ϵ , но гораздо надежнее проконтролировать этот процесс вручную. Ясно, что при очень маленьких ϵ мы получим очень маленькую невязку и не слишком гладкое решение (случай $\epsilon = 0$ соответствует полному отсутствию регуляризации). Напротив, при достаточно большом ϵ мы получим весьма гладкое решение, но с большой невязкой. Иными словами, полученный результат (хотя и гладкий) не будет хорошим решением исходной задачи. Поэтому следует искать компромисс между гладкостью и невязкой где-то посередине между этими крайними случаями. Что именно является удовлетворительным решением, зависит от конкретной задачи. Как правило, есть целое семейство таких удовлетворительных решений (с не слишком большой невязкой и довольно гладких), соответствующее отрезку $\epsilon \in [\epsilon_{\min}, \epsilon_{\max}]$. Более того, эти решения довольно похожи друг на друга (не очень сильно отличаются друг от друга). При $\epsilon < \epsilon_{\min}$ решение становится нерегулярным, а при $\epsilon > \epsilon_{\max}$ резко растет невязка.

Еще раз подчеркнем, что определять, какая невязка еще удовлетворительна и какое именно решение еще можно считать достаточно гладким, приходится в зависимости от конкретной задачи.

9. Задачи для вычислительного практикума

Задача 1

Решить задачу Коши для трехмерного движения трех тел:

$$\begin{aligned} m_1 \ddot{\vec{x}}_1 &= -g_{12}(\vec{x}_1 - \vec{x}_2)|\vec{x}_1 - \vec{x}_2|^{3/7} - g_{13}(\vec{x}_1 - \vec{x}_3)|\vec{x}_1 - \vec{x}_3|^{3/7}, \\ m_2 \ddot{\vec{x}}_2 &= -g_{12}(\vec{x}_2 - \vec{x}_1)|\vec{x}_2 - \vec{x}_1|^{3/7} - g_{23}(\vec{x}_2 - \vec{x}_3)|\vec{x}_2 - \vec{x}_3|^{3/7}, \\ m_3 \ddot{\vec{x}}_3 &= -g_{13}(\vec{x}_3 - \vec{x}_1)|\vec{x}_3 - \vec{x}_1|^{3/7} - g_{23}(\vec{x}_3 - \vec{x}_2)|\vec{x}_3 - \vec{x}_2|^{3/7}. \end{aligned}$$

Значения констант $m_1, m_2, m_3, g_{12}, g_{13}, g_{23}$ и начальные данные $\vec{x}_1, \vec{x}_2, \vec{x}_3, \dot{\vec{x}}_1, \dot{\vec{x}}_2, \dot{\vec{x}}_3$ приведены во входном файле. Найти координаты частиц в момент времени $t = 250$. Добейтесь глобальной абсолютной точности не хуже $1.0 \cdot 10^{-9}$ (при изменении локальной точности в 10 раз ответ должен меняться не более чем на $1.0 \cdot 10^{-9}$). Используйте интерполяционный метод и метод Рунге-Кутты. Сравните их скорости (при данной точности).

В этой задаче особого физического смысла нет, но оценить эффективность соответствующих методов она позволяет. Здесь надо обратить внимание на то, чтобы студенты находили правую часть ОДУ разумным образом, т. е. не вычисляли по два раза одно и то же.

Если уровень подготовки студентов невысок, то можно отказаться от сравнения методов, тогда будет два варианта задачи — с интерполяционным методом и с методом Рунге-Кутты.

Задача 2

Найти уровни энергии квантовомеханической частицы в потенциале:

$$\begin{aligned} \psi''(x) + (E - V(x))\psi(x) &= 0, \\ V(x) &= -\alpha * \frac{1.0 - 0.8 * \cos(\beta * (x - 0.01))}{\cos h(\gamma * x)}, \end{aligned}$$

где $\alpha = 20, \beta = 0.41, \gamma = 0.42$. Найти 6 нижних уровней энергии, начиная с основного состояния E_1 . Точность определения уровней должна быть не менее 10^{-11} .

Эта сравнительно несложная задача имеет непосредственный физический смысл. Физик-теоретик безусловно должен уметь искать дискретные

уровни для уравнения Шредингера. Кроме того, эта задача позволяет напомнить свойства спектра в двойной яме.

Задача 3

Найти 3 нижних p -уровня энергии квантовомеханической частицы в сферически-симметричном потенциале:

$$\psi''(r) + (E - V(r))\psi(r) = 0,$$

$$V(r) = -\frac{\alpha}{r} + \frac{l(l+1)}{r^2} + \frac{\beta}{r^4},$$

$$\text{для } \alpha = 1, \quad l = 1, \quad \beta = 0.0$$

$$\text{и для } \alpha = 1, \quad l = 1, \quad \beta = 0.001.$$

Точность определения уровней должна быть не менее 10^{-10} .

Эта задача про особую точку в радиальном уравнении Шредингера. Случай $\beta = 0.0$ позволяет сравнить численный ответ с точным. Теоретики безусловно должны уметь находить уровни энергии атома водорода.

Задача 4

Дана система двух уравнений типа Шредингера:

$$\psi_1'' + \left(E + \frac{2.2}{r} - \frac{2.0 \cdot 10^{-10}}{r^4} \right) \psi_1 + \left(\frac{0.2}{r} \right) \psi_2 = 0,$$

$$\psi_2'' + \left(E + \frac{2.2}{r} + \frac{\xi}{r^3} - \frac{2.0 \cdot 10^{-10}}{r^4} \right) \psi_2 + \left(\frac{0.2}{r} \right) \psi_1 = 0.$$

Для каждого из значений $\xi = 0.0$, $\xi = 1.21 \cdot 10^{-5}$, $\xi = 1.31 \cdot 10^{-5}$, $\xi = 1.401 \cdot 10^{-5}$, $\xi = 1.413 \cdot 10^{-5}$ надо найти по 4 нижних уровня энергии.

Это, в общем-то, излишне усложненная задача. Здесь и особая точка, и две компоненты волновой функции в уравнении Шредингера. (Впрочем, это попросту спиновые компоненты из уравнения Паули). Зато это настоящая задача «из жизни физика-теоретика». Кроме того, здесь хорошо виден эффект «завала» уровней энергии в более узкую часть потенциальной ямы при возрастании коэффициента ξ , так что физический смысл в этой задаче есть.

Задача 5

Решить задачу Коши для уравнения Шредингера

$$i\partial_0\psi(x,t) = -\partial_1^2\psi(x,t) + V(x)\psi(x,t),$$

$$V(x) = U_0\delta(x+a) + U_0\delta(x-a).$$

Начальные условия выбрать в виде гауссового пакета, причем средняя энергия должна совпадать с энергией главного резонанса. Подобрать диапазон изменения координаты x и диапазон изменения времени t так, чтобы можно было проследить всю картину рассеяния пакета. Подобрать ширину пакета и величину потенциала так, чтобы были видны колебания метастабильного уровня и экспоненциальные хвосты пакетов после рассеяния.

Это полуаналитическая-полусчетная задача, в которой главным является правильная «подгонка» параметров. Поэтому давать ее можно только при очень высоком уровне подготовки студентов. Тем не менее (при благоприятном исходе), эта задача очень полезна для лучшего понимания процессов рассеяния в квантовой механике.

Задача 6

Решить задачу Коши для уравнения

$$\partial_0^2 u(x,t) = \partial_1^2 u(x,t) - \sin(u(x,t))$$

на решетке $x_1 \dots x_{10000}$ с шагом $dx = 0.005$. Начальные данные $u(x_i, t_0)$ и $\partial_0 u(x_i, t_0)$ при $t_0 = -72.28593$ приведены во входном файле. Найти $u(x_i, t_1)$ при $t_1 = 72.28593$.

Разумеется, здесь решается задача, для которой известен точный аналитический ответ.

Прекрасно известны решения для рассеяния пары солитон-солитон:

$$4 \operatorname{arctg} \left(\frac{v \operatorname{sh} (x/\sqrt{1-v^2})}{\operatorname{ch} (vt/\sqrt{1-v^2})} \right),$$

для рассеяния пары солитон-антисолитон:

$$4 \operatorname{arctg} \left(\frac{\operatorname{sh} (vt/\sqrt{1-v^2})}{v \operatorname{ch} (x/\sqrt{1-v^2})} \right)$$

и для периодического решения (бризера):

$$4 \operatorname{arctg} \left(\frac{\sin(vt/\sqrt{1+v^2})}{v \operatorname{ch}(x/\sqrt{1+v^2})} \right).$$

Надо попросту подобрать подходящую скорость v так, чтобы за разумный временной интервал $[t_0, t_1]$ происходил полный цикл рассеяния или полный цикл эволюции бризера. При этом скорость не должна быть ни слишком маленькой — тогда потребуется слишком много времени, ни слишком большой — тогда упадет точность счета.

Задача 7

Решить уравнение

$$-\Delta\varphi(x, y) + \varphi(x, y) + 2\varphi^3(x, y) = 0$$

в квадрате $0 \leq x \leq 1, 0 \leq y \leq 1$ с граничными условиями:

$$\varphi(x, 0) = \sin(\pi \cdot x), \quad \varphi(x, 1) = \sin(2\pi \cdot x),$$

$$\varphi(0, y) = \sin(\pi \cdot y), \quad \varphi(1, y) = \sin(2\pi \cdot y).$$

Использовать равномерную сетку 301×301 , $dx = dy = 1/300$, $x_i = i * dx, y_j = j * dy, i, j = 0 \dots 300$. Невязка во всех узлах сетки не должна превышать $1.0 \cdot 10^{-8}$. Выходной файл должен содержать значения $\varphi(x_{150}, y_j), j = 0 \dots 300$. Используйте чебышевское ускорение и метод минимизации невязки. Сравните их скорости.

Вообще, это чисто иллюстративная задача, особого смысла у нее нет. Если уровень подготовки студентов невысок, то можно отказаться от сравнения методов, тогда получится два варианта задачи — с классическим чебышевским методом и с методом минимизации невязки.

Задача 8

Найти основное состояние (дискретный уровень с наименьшей энергией) в прямоугольной потенциальной яме для уравнения:

$$-\left(\psi(x+i) + \psi(x-i) - 2\psi(x)\right) + \left(E - V(x)\right)\psi(x) = 0,$$

$$V(x) = -V_0\theta(a - |x|).$$

Чтобы найти уровень, решите интегральное уравнение

$$\psi(x) = \int dy \, g(k, x - y) V(y) \psi(y),$$

здесь

$$g(k, x) = - \frac{1}{\sin k} \frac{\operatorname{sh}((\pi - k)x)}{\operatorname{sh}((\pi - 0)x)},$$

где $E = 2 \cos k - 2$.

Это настоящая задача «из жизни физика-теоретика». При высоком уровне подготовки студентов можно не указывать метод решения задачи через интегральное уравнение — пусть сначала попробуют продолжить функцию на комплексную плоскость с помощью интерполяции (достаточной точности при этом не получится).

Литература

Общий для всего курса численных методов список рекомендованной литературы и комментарии к нему мы уже привели в части I. Здесь мы приводим этот список еще раз для удобства читателей.

- [1] Н. Н. Калиткин «Численные методы». — М.: «Наука», 1978.
- [2] Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков «Численные методы». — М.: «Наука», 1973; М.: «Наука», 1987.
- [3] А. А. Самарский «Введение в численные методы». — М.: «Наука», 1982; М.: «Наука», 1987; М.: «Наука», 1997.
- [4] Л. И. Турчак «Основы численных методов». — М.: «Наука», 1987.
- [5] В. И. Крылов, В. В. Бобков, П. И. Монастырский «Начала теории вычислительных методов». — Минск: «Наука и техника», 1986.
- [6] Е. А. Волков «Численные методы». — М.: «Наука», 1982.
- [7] В. И. Приклонский «Численные методы в физике». — М.: МГУ, 1999.
- [8] В. И. Косарев «12 лекций по вычислительной математике». — М.: МФТИ, 1995.
- [9] М. К. Гавурин «Лекции по методам вычислений». — М.: «Наука», 1971.
- [10] А. А. Самарский «Введение в теорию разностных схем». — М.: «Наука», 1971.
- [11] А. А. Самарский, А. В. Гулин «Численные методы математической физики». — М.: «Научный мир», 2000.
- [12] А. Н. Тихонов, В. Я. Арсенин «Методы решения некорректных задач». — М.: «Наука», 1986.

- [13] Р.Рихтмайер, К.Мортон «Разностные методы решения краевых задач». — М.: «Мир», 1972.
- [14] Stoer J., Bulirsh R. «Introduction to numerical analysis». — New-York: Springer-Verlag, 1980.
- [15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery «Numerical recipes in C». — Cambridge University Press, 1995.
- [16] F. S. Acton «Numerical methods that work». — Washington: Mathematical association of America, 1990.
- [17] D. E. Knuth «The art of computer programming». — Reading, MA: Addison-Wesley, 1989.
- [18] G. H. Golub, C. F. Van Loan «Matrix computations». — Baltimore: John Hopkins University Press, 1989.

**Ильина Вера Алексеевна
Силаев Петр Константинович**

**ЧИСЛЕННЫЕ МЕТОДЫ ДЛЯ ФИЗИКОВ-ТЕОРЕТИКОВ
Часть 2**

*Дизайнер М. В. Ботя
Технический редактор А. В. Ширококов
Компьютерная верстка Ю. С. Кирьянова
Корректор Н. В. Сухих*

Подписано в печать 09.02.04. Формат 60 × 84¹/₁₆. Печать офсетная.
Усл. печ. л. 6,86. Уч. изд. л. 6,43. Гарнитура Таймс. Бумага офсетная №1. Заказ №177.

АНО «Институт компьютерных исследований»
426034, г. Ижевск, ул. Университетская, 1.

Лицензия на издательскую деятельность ЛУ №084 от 03.04.00.
<http://red.ru> E-mail: borisov@red.ru
